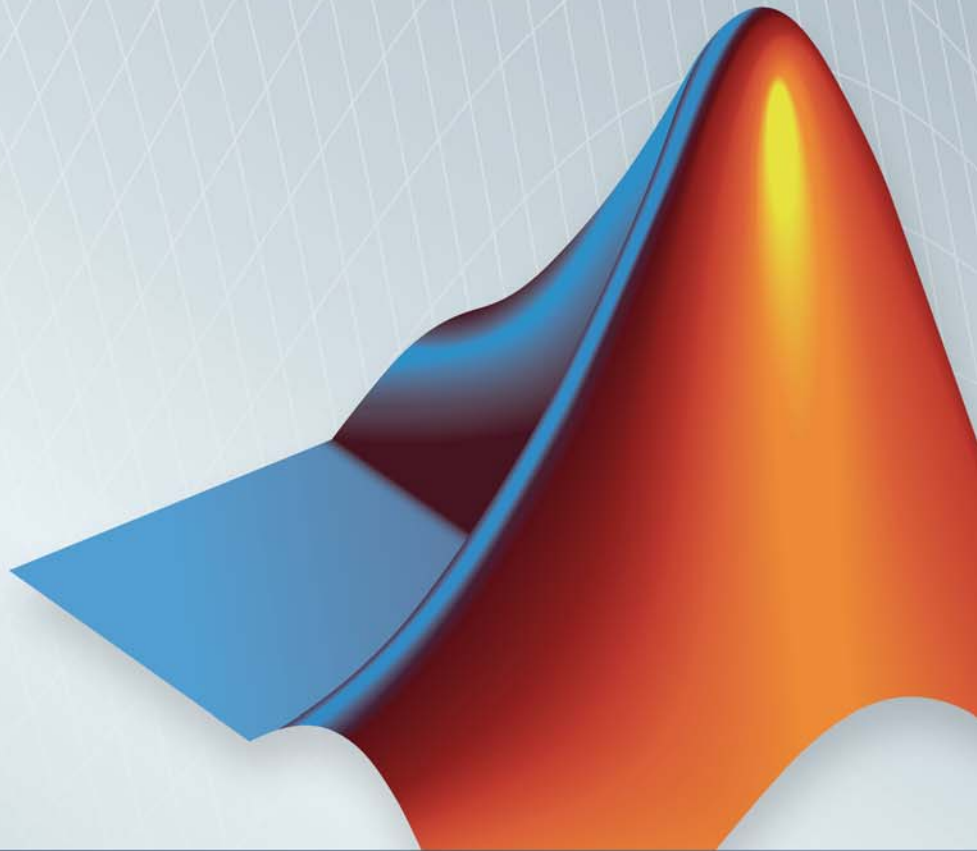


**xPC Target™**

Reference

**R2013a**



**MATLAB® & SIMULINK®**



## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com)  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab)  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html)

Web  
Newsgroup  
Technical Support



[suggest@mathworks.com](mailto:suggest@mathworks.com)  
[bugs@mathworks.com](mailto:bugs@mathworks.com)  
[doc@mathworks.com](mailto:doc@mathworks.com)  
[service@mathworks.com](mailto:service@mathworks.com)  
[info@mathworks.com](mailto:info@mathworks.com)

Product enhancement suggestions  
Bug reports  
Documentation error reports  
Order status, license renewals, passcodes  
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

*xPC Target™ Reference*

© COPYRIGHT 2002–2013 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

**Revision History**

March 2007	Online only	New for Version 3.2 (Release 2007a)
September 2007	Online only	Updated for Version 3.3 (Release 2007b)
March 2008	Online only	Updated for Version 3.4 (Release 2008a)
October 2008	Online only	Updated for Version 4.0 (Release 2008b)
March 2009	Online only	Updated for Version 4.1 (Release 2009a)
September 2009	Online only	Updated for Version 4.2 (Release 2009b)
March 2010	Online only	Updated for Version 4.3 (Release 2010a)
April 2011	Online only	Updated for Version 5.0 (Release 2011a)
September 2011	Online only	Updated for Version 5.1 (Release 2011b)
March 2012	Online only	Revised for Version 5.2 (Release 2012a)
September 2012	Online only	Revised for Version 5.3 (Release 2012b)
March 2013	Online only	Revised for Version 5.4 (Release 2013a)



## Functions

**1**

## xPC Target API Reference for C

**2**

<b>C API Functions</b> .....	2-2
Target Computers .....	2-2
Target Applications .....	2-3
Scopes .....	2-4
Parameters .....	2-6
Signals .....	2-7
Data Logs .....	2-7
File Systems .....	2-8
Errors .....	2-9
<b>C API Error Messages</b> .....	2-10
<b>C API Structures and Functions — Alphabetical List</b> ..	2-14

## xPC Target API Reference for COM

**3**

<b>COM API Methods</b> .....	3-2
Target Computers .....	3-2
Target Applications .....	3-3
Scopes .....	3-4
Parameters .....	3-6
Signals .....	3-6
Data Logs .....	3-7
File Systems .....	3-7

Errors .....	3-8
COM API Methods — Alphabetical List .....	3-9

## Configuration Parameters

# 4

<b>Setting Configuration Parameters .....</b>	<b>4-2</b>
xPC Target options Pane .....	4-3
Automatically download application after building .....	4-4
Download to default target PC .....	4-5
Specify target PC name .....	4-6
Name of xPC Target object created by build process .....	4-7
Use default communication timeout .....	4-8
Specify the communication timeout in seconds .....	4-9
Execution mode .....	4-10
Real-time interrupt source .....	4-11
I/O board generating the interrupt .....	4-12
PCI slot (-1: autosearch) or ISA base address .....	4-16
Log Task Execution Time .....	4-17
Signal logging data buffer size in doubles .....	4-18
Enable profiling .....	4-20
Number of events (each uses 20 bytes) .....	4-21
Double buffer parameter changes .....	4-22
Load a parameter set from a file on the designated target file system .....	4-24
File name .....	4-25
Build COM objects from tagged signals/parameters .....	4-26
Generate CANape extensions .....	4-27
Include model hierarchy on the target application .....	4-28
Enable Stateflow animation .....	4-29

# Target Computer Command-Line Interface Reference

## 5

<b>Target Computer Commands</b> .....	<b>5-2</b>
Introduction .....	<b>5-2</b>
Target Object Methods .....	<b>5-2</b>
Target Object Property Commands .....	<b>5-3</b>
Scope Object Methods .....	<b>5-5</b>
Scope Object Property Commands .....	<b>5-7</b>
Aliasing with Variable Commands .....	<b>5-8</b>





# Functions

---

# fc422mexcalcbits

---

**Purpose** Calculate parameter values for Fastcom 422/2-PCI board

**Syntax** [a b] = fc422mexcalcbits(frequency)  
[a b df] = fc422mexcalcbits(frequency)

**Arguments** frequency Desired baud rate for the board


[a b] = fc422mexcalcbits(frequency) accepts a baud rate (in units of baud/second) and converts this value into two parameters a b. You must enter these values for the parameter **Clock bits** of the Fastcom® 422/2-PCI driver clock. The desired baud rate (frequency) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

[a b df] = fc422mexcalcbits(frequency) accepts a baud rate (in units of baud/second) and converts this value into two parameters a b. You must enter these values for the parameter **Clock bits** of the Fastcom 422/2-PCI driver block. The third value, df, indicates the actual baud rate that is created by the generated parameters a b. The clock circuit has limited resolution and is unable to perfectly match an arbitrary frequency. The desired baud rate (frequency) must range between 30e3 and 1.5e6, which is a hardware limitation of the clock circuit.

<b>Purpose</b>	List environment properties assigned to MATLAB variable
<b>Syntax</b>	<code>getxpcenv</code> <code>getxpcenv propertyname</code>
<b>Description</b>	<p><code>getxpcenv</code> displays, in the MATLAB® Command Window, the property names and current property values for the xPC Target™ environment.</p> <p><code>getxpcenv propertyname</code> displays the current value of property <code>propertyname</code>. The environment properties define communication between the host computer and target computer and the type of target boot kernel created during the setup process.</p>

---

**Tip** To access a subset of these properties in xPC Target Explorer:

- 1 Expand a target computer node in the **Targets** pane.
- 2 Click the Target Properties icon  in the toolbar or double-click **Properties**.

- 
- “Host-to-Target Communication” on page 1-4
  - “Target Settings” on page 1-10
  - “Boot Configuration” on page 1-13
  - “Host Configuration” on page 1-16

## Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.</p> <hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>

Environment Property	Description
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to 255.255.255.255, which means that a gateway is not used to connect to the target computer.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>

Environment Property	Description
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p>

Environment Property	Description
	<p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>

Environment Property	Description
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.</p>



Environment Property	Description
TcpIpTargetISAMemPort	<p>Property value is '0xnxxx'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area</p>

Environment Property	Description
	(telnet, ftp, . . .) and is only of use on the target computer.

## Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.</p>
MaxModelSize	<p>Property values are '1MB' (the default), '4MB', and '16MB'.</p> <p>Select 1 MB, 4 MB, or 16 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.</p>

Environment Property	Description
	<p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.</p> <ul style="list-style-type: none"> <li>• BootFloppy and DOSLoader modes ignore this value.</li> <li>• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.</li> </ul>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.</p>
Name	Target computer name.
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.</p>

Environment Property	Description
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of xPC Target Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS,</p>

Environment Property	Description
	<p>you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.</p> <p>The xPC Target kernel can use only 2 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to 'off'.</p>

### Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.

Environment Property	Description
EmbeddedOption	<p>Property values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option™ product is enabled only if you purchase an additional license.</p>
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.</p>

Environment Property	Description
	<p><b>Tip</b> In the <b>Target Properties</b> pane of xPC Target Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p> <p>xx:xx:xx:xx:xx:xx</p> <p>To update the MAC address in xPC Target Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.</p>

## Host Configuration

Environment Property	Description
Version	xPC Target version number. Displayed only from <code>getxpcenv</code> when called without arguments.

## Examples

Return the xPC Target environment in the structure shown below. The output in the MATLAB window is suppressed. The structure contains three fields for property names, current property values, and new property values.

```
env = getxpcenv
env =
    propname: {1x25 cell}
    actpropval: {1x25 cell}
    newpropval: {1x25 cell}
```

Display a list of the environment property names, current values, and new values.

```
env = getxpcenv
```

## See Also

`setxpcenv` | `xpcbootdisk`



<b>Purpose</b>	Retrieve diagnostic information to help troubleshoot configuration issues		
<b>Syntax</b>	<code>getxpcinfo</code> <code>getxpcinfo('-a')</code>		
<b>Arguments</b>	<table><tr><td><code>'-a'</code></td><td>Appends diagnostic information to an existing <code>xpcinfo.txt</code> file. If one does not exist, this function creates the file in the current folder.</td></tr></table>	<code>'-a'</code>	Appends diagnostic information to an existing <code>xpcinfo.txt</code> file. If one does not exist, this function creates the file in the current folder.
<code>'-a'</code>	Appends diagnostic information to an existing <code>xpcinfo.txt</code> file. If one does not exist, this function creates the file in the current folder.		

**Description** `getxpcinfo` returns diagnostic information for troubleshooting xPC Target configuration issues. This function generates and saves the information in the `xpcinfo.txt` file, in the current folder. If the file `xpcinfo.txt` already exists, this function overwrites it with the new information.

`getxpcinfo('-a')` appends the diagnostic information to the `xpcinfo.txt` file, in the current folder. If the file `xpcinfo.txt` does not exist, this function creates it.

You can send the file `xpcinfo.txt` to MathWorks® Technical Support for evaluation and guidance. To create this file, you must have write permission for the current folder.

### Warning

**The file `xpcinfo.txt` might contain information sensitive to your organization. Review the contents of this file before sending to MathWorks.**

# macaddr

---

**Purpose** Convert string-based MAC address to vector-based one

**Syntax** `macaddr('MAC address')`

**Argument** 'MAC address' String-based MAC address to be converted.

**Description** `macaddr('MAC address')` converts a string-based MAC address to a vector-based MAC address. The string-based MAC address should be a string comprised of six colon-delimited fields of two-digit hexadecimal numbers.

**Examples** `macaddr('01:23:45:67:89:ab')`

`ans =`

`1 35 69 103 137 171`

**How To** • “Model-Based Ethernet Communications”

**Purpose**

Change xPC Target environment properties

**Syntax**

```
setxpcenv
setxpcenv('property_name','property_value')
setxpcenv('prop_name1','prop_value1','prop_name2',. . .)
```

**Arguments**

**property\_name** Not case sensitive. Property names can be shortened as long as they can be differentiated from the other property names.

**property\_value** Character string. Type `setxpcenv` without arguments to get a listing of allowed values. Property values are not case sensitive.

**Description**

Function to enter new values for environment properties. If the new value is different from the current value, the property is marked as having a new value. `setxpcenv` works similarly to the `set` function of the MATLAB Handle Graphics® system.

`setxpcenv` called without arguments returns a list of allowed property values in the MATLAB window.


`setxpcenv('property_name','property_value')` sets property `property_name` to `property_value`.

`setxpcenv('prop_name1','prop_value1','prop_name2',. . .)` is called with one or more argument pairs. The first argument of a pair is the property name; the second is the new value for this property.

The environment properties define communication between the host computer and target computer and the type of target boot kernel created during the setup process. With the exception of the `Version` property, you can set environment properties using the `setxpcenv` function or the xPC Target Explorer window, accessed via the `xpcexplr` function. An understanding of the environment properties will help you configure the xPC Target environment.

---

**Tip** To access a subset of these properties in xPC Target Explorer:

- 1 Expand a target computer node in the **Targets** pane.
- 2 Click the Target Properties icon  in the toolbar or double-click **Properties**.

- 
- “Host-to-Target Communication” on page 1-20
  - “Target Settings” on page 1-26
  - “Boot Configuration” on page 1-30

## Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.</p>

Environment Property	Description
	<p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p>
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>

Environment Property	Description
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to 255.255.255.255, which means that a gateway is not used to connect to the target computer.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>

Environment Property	Description
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>

Environment Property	Description
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings</p>



Environment Property	Description
	<p>leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.</p>
TcpIpTargetISAMemPort	<p>Property value is '0xnnnn'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the</p>

Environment Property	Description
	<p><b>Target Properties</b> pane of xPC Target Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer.</p>

## Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.</p>

Environment Property	Description
MaxModelSize	<p>Property values are '1MB' (the default), '4MB', and '16MB'.</p> <p>Select 1 MB, 4 MB, or 16 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.</p> <ul style="list-style-type: none"> <li>• BootFloppy and DOSLoader modes ignore this value.</li> <li>• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.</li> </ul>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.</p>
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of xPC Target Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.</p> <p>The xPC Target kernel can use only 2 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p>

Environment Property	Description
	Set this value to 'on' if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to 'off'.

## Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are 'Disabled' and 'Enabled'. This property is read only.  Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

Environment Property	Description
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of xPC Target Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p> <hr/>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p>

Environment Property	Description
	<p>xx:xx:xx:xx:xx:xx</p> <p>To update the MAC address in xPC Target Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.</p>

## Examples

List the current environment properties.

```
setxpcenv
```

Change the serial communication port of the host computer to COM2.

```
setxpcenv('RS232HostPort','COM2')
```

## See Also

```
getxpcenv | xpcbootdisk
```

## How To

- “Ethernet Communication Setup”
- “RS-232 Communication Setup”
- “Target Boot Methods”
- “Command-Line Setup”



**Purpose**

Benchmark xPC Target models on target computer

**Syntax**

```
xpcbench
xpcbench benchmark
xpcbench benchmark -reboot
xpcbench benchmark -cleanup
xpcbench benchmark -verbose
xpcbench benchmark -reboot -cleanup -verbose

expected_results = xpcbench()
current_results = xpcbench(benchmark, ___)
```

**Description**

xpcbench benchmarks the real-time execution performance of xPC Target applications on your target computer. It compares the result to stored benchmark results from other computers.

Benchmark execution includes generating benchmark models, building and downloading xPC Target applications, searching for the minimal achievable sample time, and displaying results.

xpcbench without an argument displays two plots containing representative results for five benchmarks run on various target computers.

- For each target computer tested, the first plot lists the minimal achievable sample time for the five benchmarks, in microseconds.
- The second plot contains a bar graph of all computers tested, ranked by relative performance.

The results are labeled by CPU type, CPU clock rate, and the compiler for the application.

Depending upon the value of benchmark, xpcbench benchmark produces different outputs:

- `xpcbench` this displays two plots containing benchmark results for the five benchmarks run on your target computer, compared with the representative benchmark results for other target computers:
  - For each target computer tested, the first plot lists the minimal achievable sample time for the benchmarks, in microseconds.
  - The second plot contains a bar graph of all computers tested, ranked by relative performance.

The results are labeled by CPU type, CPU clock rate, and the compiler used to compile the application. The entry for your target computer is highlighted.

- `xpcbench benchmark` prints the benchmark name, the number of blocks, the model build time in seconds, the execution time in seconds, and the minimal achievable sample time in microseconds in the MATLAB command window.

`xpcbench benchmark -reboot` runs the benchmark, then restarts the target computer.

`xpcbench benchmark -cleanup` runs the benchmark, plots or prints benchmark results, and deletes the build files.

`xpcbench benchmark -verbose` prints build output, runs the benchmark, and plots or prints benchmark results.

`xpcbench benchmark -reboot -cleanup -verbose` prints build output, restarts the target computer, deletes build files, and plots or prints results.

You can add zero or more of these control arguments in arbitrary order.

`expected_results = xpcbench()` returns the benchmark results for the five predefined benchmarks in a structure array.

Depending upon the value of `benchmark`, `current_results = xpcbench(benchmark, ___)` returns different results:

- `xpcbench('this')` returns the benchmark results for the five predefined benchmarks in a structure array.
- `xpcbench(benchmark)` returns the benchmark results for the specified model in a structure.

## Input Arguments

### **benchmark - Benchmark name or model name**

`this` | `usermdl` | `minimal` | `f14` | `f14*5` | `f14*10` | `f14*25`

Benchmark, specified as a literal string or string variable containing one of:

<code>this</code>	All five predefined benchmark models ( <code>minimal</code> , <code>f14</code> , <code>f14*5</code> , <code>f14*10</code> , <code>f14*25</code> )
<code>usermdl</code>	Your model, <code>usermdl</code> .
<code>minimal</code>	Minimal model consisting of three blocks (Constant, Gain, Termination).
<code>f14</code>	Standard Simulink® example <code>f14</code> (62 blocks, 10 continuous states).
<code>f14*5</code>	Five <code>f14</code> systems modeled in subsystems (310 blocks, 50 continuous states).
<code>f14*10</code>	Ten <code>f14</code> systems (620 blocks, 100 continuous states).
<code>f14*25</code>	25 <code>f14</code> systems (1550 blocks, 250 continuous states).

When using function form, enclose literal arguments (`this`, `-reboot`) in single quotes (`'this'`, `'-reboot'`).

**Example:**

## Output Arguments

### Data Types

char

### **expected\_results - Results of predefined benchmarks previously run on representative target computers**

struct array

xpcbench without an argument returns representative benchmark results in a structure array with element fields:

<i>Machine</i>	Target computer information string containing CPU type, CPU speed, compiler
<i>BenchResults</i>	Target computer benchmark performance for all five predefined benchmarks
<i>Desc</i>	Target computer descriptor string containing machine type, RAM size, cache size

### **current\_results - Current results of specified benchmark**

struct

xpcbench(benchmark) returns the benchmark results in a structure with fields:

<i>Name</i>	Benchmark name
<i>nBlocks</i>	Number of blocks in benchmark
<i>BuildTime</i>	Elapsed time in seconds to build benchmark
<i>BenchTime</i>	Elapsed time in seconds to run benchmark
<i>Tsmin</i>	Minimal achievable sample time in seconds for benchmark

## Tips

- Before you run `xpcbench`, you must be able to start the target computer, connect the host computer to the target computer, and run the confidence test, `xpctest`, with no failures.
- After running `xpcbench` on your model and system, set your model sample time to the minimal achievable sample time value reported. Smaller sample times overload the target computer.
- The stored benchmark results were collected with **Multicore CPU support** disabled. When evaluating your system, temporarily disable this target setting using `xpcexplr`.
- The stored benchmark models were compiled using a sampling of the currently supported compilers. When evaluating your system, find the closest match to the compiler you are using.
- Benchmark `minimal` has neither continuous nor discrete states. It provides information about the target computer interrupt latencies.

## Examples

### `xpcbench`

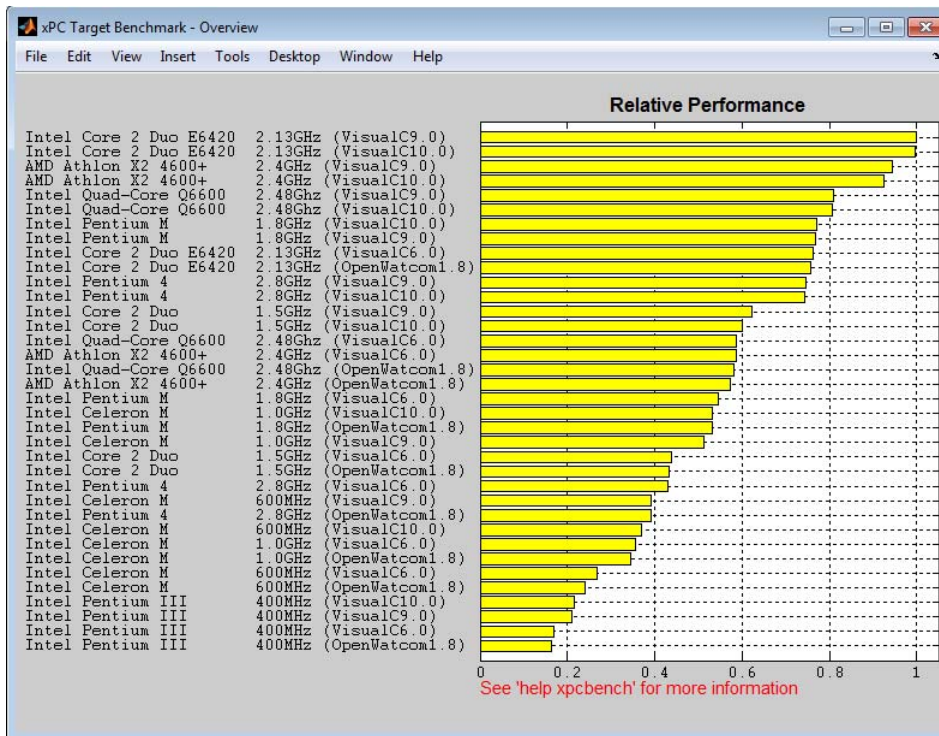
Show representative benchmark results from various target computers.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Display representative results on five predefined benchmarks.

```
xpcbench
```



Minimal achievable sample times in  $\mu$ s

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

### xpcbench this

Use the five predefined benchmarks to benchmark the target computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run five benchmark models and display results.

```
xpcbench this
```

```
### Starting xPC Target build procedure for model:
    xpcminimal
### Successful completion of build procedure for model:
```

```
    xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

### Starting xPC Target build procedure for model:
    f14tmp1
### Successful completion of build procedure for model:
    f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp1

### Starting xPC Target build procedure for model:
    f14tmp5
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5

### Starting xPC Target build procedure for model:
    f14tmp10
### Successful completion of build procedure for model:
    f14tmp10
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp10

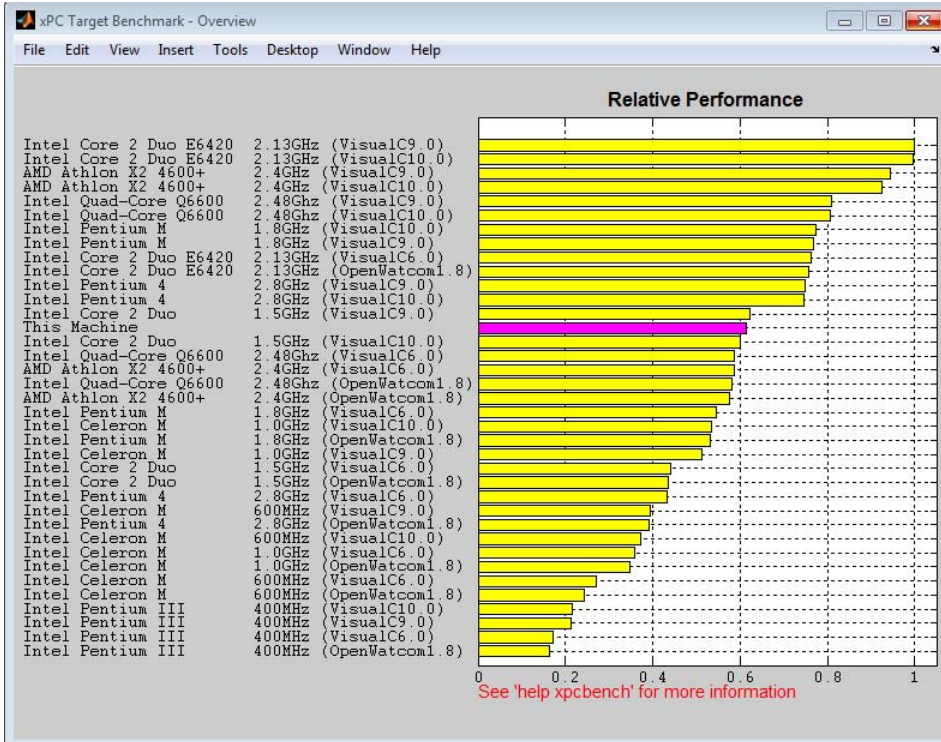
### Starting xPC Target build procedure for model:
    f14tmp25
### Successful completion of build procedure for model:
    f14tmp25
```



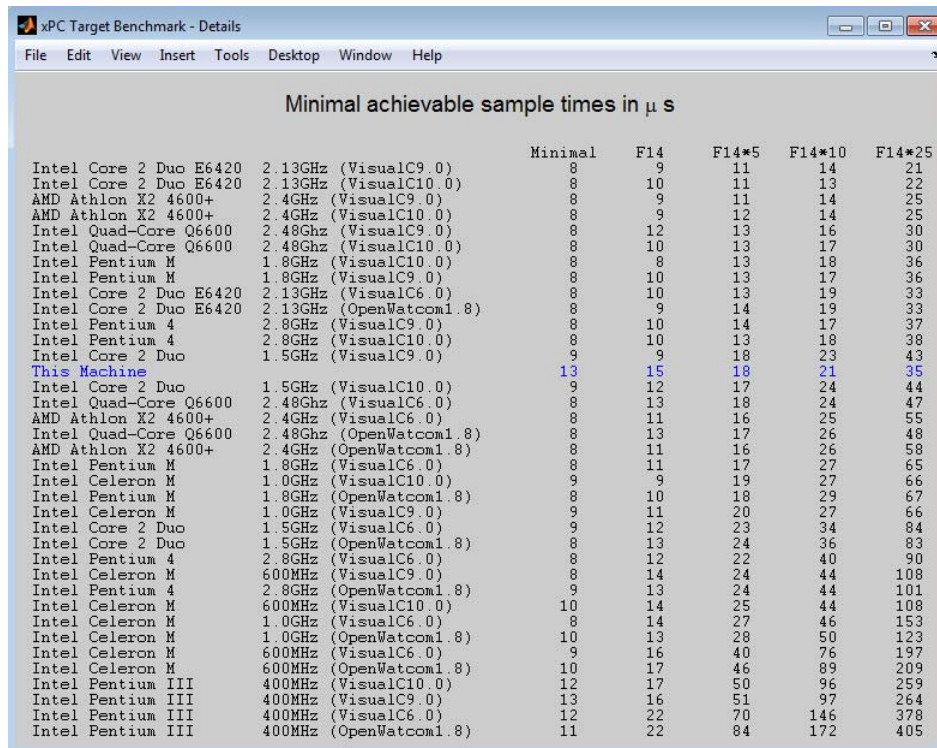
```

### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25
    
```



# xpcbench



Minimal achievable sample times in  $\mu$ s

	Minimal	F14	F14*5	F14*10	F14*25	
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
<u>This Machine</u>		13	15	18	21	35
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48GHz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48GHz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

## xpcbench xpcosc

Use model xpcosc to benchmark the target computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run benchmark on xpcosc and print results.

```
xpcbench xpcosc
```

```
### Starting xPC Target build procedure for model:  
xpcosc
```

```
### Successful completion of build procedure for model:
      xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcosc

Benchmark results for model:                xpcosc
Number of blocks in model:                  10
Elapsed time for model build (sec):         25.9
Elapsed time for model benchmark (sec):     23.4
Minimal achievable sample time (microsec): 11.9
```

### **xpcbench minimal**

Use the minimal model to benchmark the target computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run benchmark on minimal model and print results.

```
xpcbench minimal
```

```
### Starting xPC Target build procedure for model:
      xpcminimal
### Generated code for 'xpcminimal' is up to date
### Successful completion of build procedure for model:
      xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

Benchmark results for model:                Minimal
Number of blocks in model:                  3
Elapsed time for model build (sec):         12.8
Elapsed time for model benchmark (sec):     23.4
```

Minimal achievable sample time (microsec): 8.5

## **xpcbench f14\*5 -reboot**

Benchmark the target computer with five copies of model f14, and then restart the computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run benchmark on f14\*5, restart, and print results.

```
xpcbench f14*5 -reboot
```

```
### Starting xPC Target build procedure for model:
    f14tmp5
### Generated code for 'f14tmp5' is up to date
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5
### Reboot target: TargetPC1..... OK
```

```
Benchmark results for model:          F14*5
Number of blocks in model:           310
Elapsed time for model build (sec):   14.6
Elapsed time for model benchmark (sec): 29.7
Minimal achievable sample time (microsec): 18.1
```

## **xpcbench f14\*10 -reboot**

Benchmark the target computer with ten copies of model f14, and then restart the computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run benchmark on f14\*10, restart, and print results.

```
xpcbench f14*10 -reboot
```

```
### Starting xPC Target build procedure for model:
    f14tmp10
### Generated code for 'f14tmp10' is up to date
### Successful completion of build procedure for model:
    f14tmp10
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp10
### Reboot target: TargetPC1..... OK
```

```
Benchmark results for model:           F14*10
Number of blocks in model:             620
Elapsed time for model build (sec):    18.3
Elapsed time for model benchmark (sec): 31.9
Minimal achievable sample time (microsec): 21.4
```

### **xpcbench this -cleanup**

Benchmark the target computer with the five predefined benchmarks, and then delete build files.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run the five benchmark models, clean up build files, and display results.

```
xpcbench this -cleanup
```

```
### Starting xPC Target build procedure for model:
    xpcminimal
### Generated code for 'xpcminimal' is up to date
### Successful completion of build procedure for model:
    xpcminimal
```

```
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

### Starting xPC Target build procedure for model:
    f14tmp1
### Generated code for 'f14tmp1' is up to date
### Successful completion of build procedure for model:
    f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp1

### Starting xPC Target build procedure for model:
    f14tmp5
### Generated code for 'f14tmp5' is up to date
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5

### Starting xPC Target build procedure for model:
    f14tmp10
### Generated code for 'f14tmp10' is up to date
### Successful completion of build procedure for model:
    f14tmp10
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp10

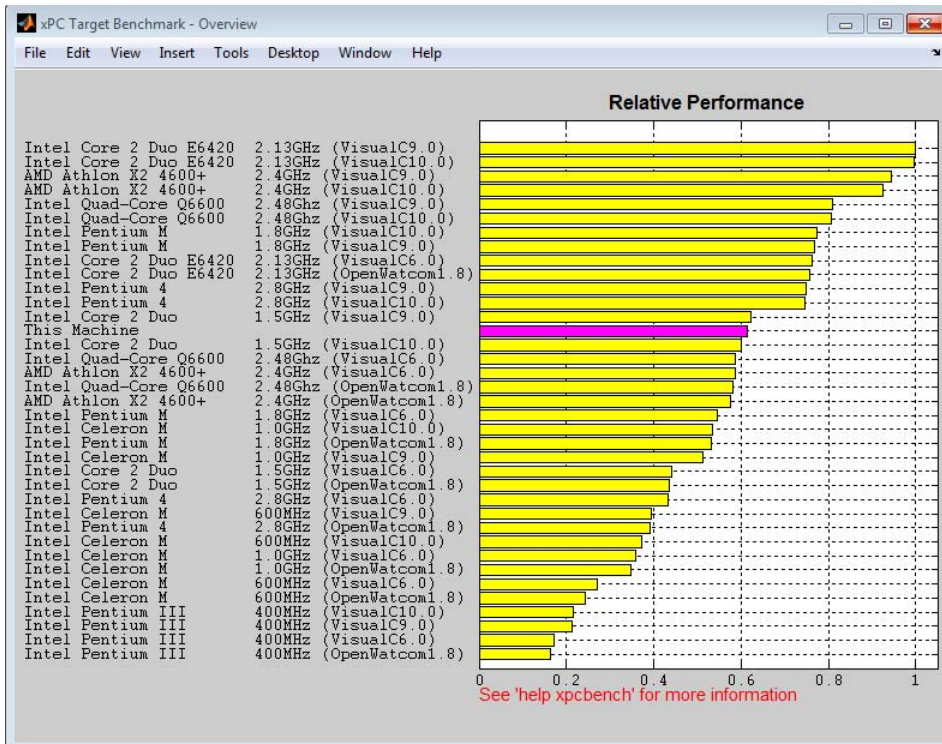
### Starting xPC Target build procedure for model:
    f14tmp25
```

```

### Generated code for 'f14tmp25' is up to date
### Successful completion of build procedure for model:
    f14tmp25
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25

```



Minimal achievable sample times in  $\mu$ s

	Minimal	F14	F14*5	F14*10	F14*25	
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48GHz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48GHz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
<b>This Machine</b>		<b>13</b>	<b>15</b>	<b>18</b>	<b>21</b>	<b>35</b>
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48GHz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48GHz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

## xpcbench xpcosc -cleanup

Benchmark the target computer with xpcosc, and then delete build files.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Run the xpcosc, clean up build files, and display results.

```
xpcbench xpcosc -cleanup
```

```
### Starting xPC Target build procedure for model:
```



```
    xpcosc
### Generated code for 'xpcosc' is up to date
### Successful completion of build procedure for model:
    xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcosc

Benchmark results for model:                xpcosc
Number of blocks in model:                  10
Elapsed time for model build (sec):         16.8
Elapsed time for model benchmark (sec):     23.4
Minimal achievable sample time (microsec): 9.9
```

### **xpcbench xpcosc -verbose**

Use xpcosc to benchmark the target computer, printing build messages.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Build the xpcosc model, print build messages, run the benchmark, and display results.

```
xpcbench xpcosc -verbose
```

```
### Starting xPC Target build procedure for model:
    xpcosc
### Generating code into build folder: xpcosc_xpc_rtw
### Invoking Target Language Compiler on xpcosc.rtw
### Using System Target File: xpctarget.tlc
### Loading TLC function libraries
.
.
.
### Created DLM ..\xpcosc.dlm
### Successful completion of build procedure for model:
```

```
        xpcosc
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create xPC Object tg

xPC Object

    Connected          = Yes
    Application        = xpcosc
    Mode               = Real-Time Single-Tasking
    Status             = stopped
    CPUOverload        = none
.
.
.
### Running benchmark for model: xpcosc

Benchmark results for model:          xpcosc
Number of blocks in model:           10
Elapsed time for model build (sec):   28.6
Elapsed time for model benchmark (sec): 23.4
Minimal achievable sample time (microsec): 14.8
```

## **xpcbench f14\*25 -cleanup -reboot**

Benchmark the target computer with 25 copies of f14, restart the target computer, and then delete build files.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Build model f14\*25, run benchmark, restart, delete build files, and print results.

```
xpcbench f14*25 -cleanup -reboot
```

```
### Starting xPC Target build procedure for model:
    f14tmp25
```

```
### Successful completion of build procedure for model:
    f14tmp25
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25
### Reboot target: TargetPC1..... OK
```

```
Benchmark results for model:                F14*25
Number of blocks in model:                  1550
Elapsed time for model build (sec):         48.8
Elapsed time for model benchmark (sec):     34.0
Minimal achievable sample time (microsec): 35.1
```

### **xpcbench f14 -verbose -cleanup -reboot**

Benchmark the target computer with model f14, print build messages, delete build files, and then restart the target computer.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Build model f14, print build messages, run benchmark, clean up build files, restart, and print results.

```
xpcbench f14 -verbose -cleanup -reboot
```

```
### Starting xPC Target build procedure for model:
    f14tmp1
### Generating code into build folder: f14tmp1_xpc_rtw
### Invoking Target Language Compiler on f14tmp1.rtw
### Using System Target File: xpctarget.tlc
### Loading TLC function libraries
.
.
.
### Created DLM ..\f14tmp1.dlm
### Successful completion of build procedure for model:
```

```
    f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create xPC Object tg

xPC Object

    Connected          = Yes
    Application        = f14tmp1
    Mode               = Real-Time Single-Tasking
    Status             = stopped
    CPUOverload       = none
.
.
.
### Running benchmark for model: f14tmp1
### Reboot target: TargetPC1..... OK

Benchmark results for model:          F14*1
Number of blocks in model:           62
Elapsed time for model build (sec):   25.3
Elapsed time for model benchmark (sec): 23.4
Minimal achievable sample time (microsec): 15.8
```

## **expected\_results = xpcbench()**

Return a structure array containing benchmark results showing what to expect of various target computers.

Start the target computer and run confidence test.

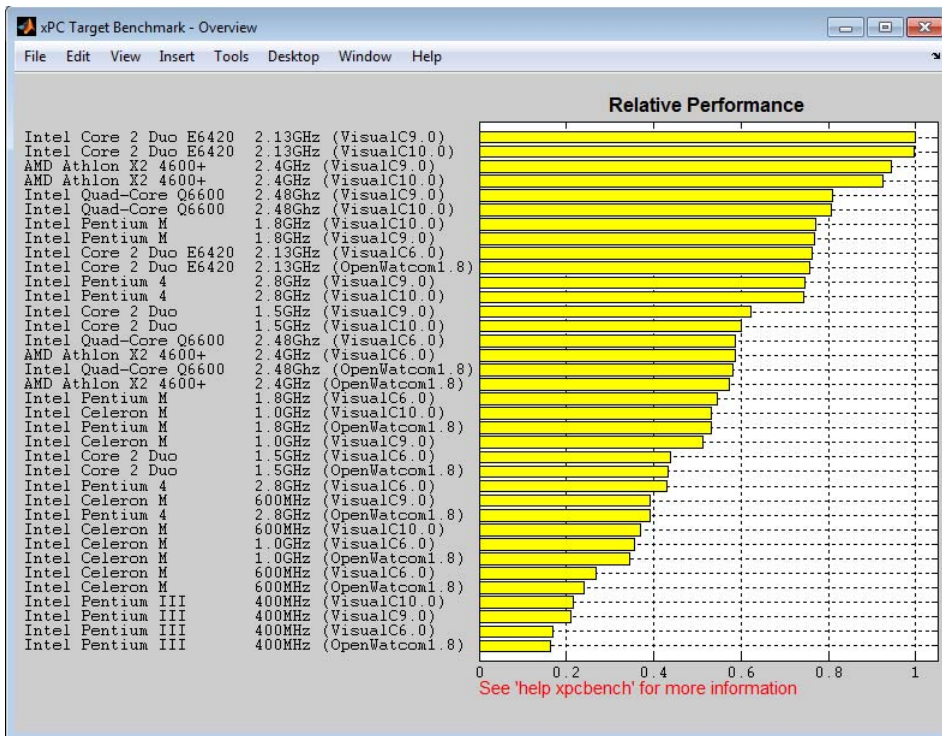
xpcbench\_prework\_snippet

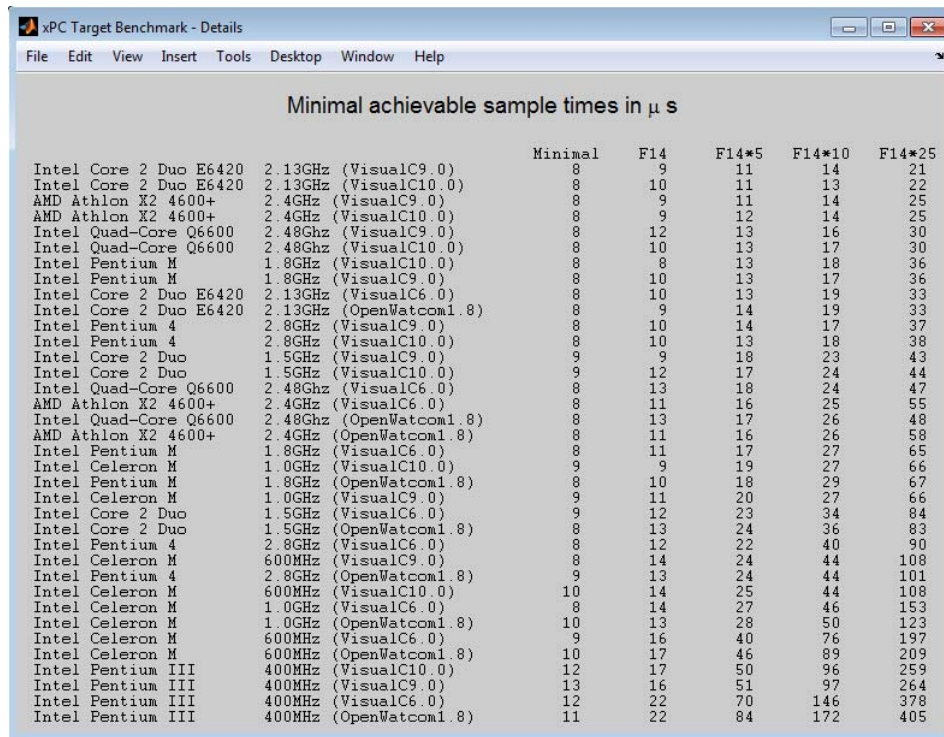
Return an array with representative results for each processor type, in arbitrary order.

```
expected_results = xpcbench();
expected_results(1)
```

ans =

```
Machine: 'Intel Celeron M
        600MHz (VisualC10.0)'
BenchResults: [1.0000e-05 1.4000e-05 2.5000e-05 4.
               4000e-05 1.0800e-04]
Desc: [1x70 char]
```





Minimal achievable sample times in  $\mu$ s

		Minimal	F14	F14*5	F14*10	F14*25
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48Ghz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48Ghz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

```
current_results = xpcbench('this','-cleanup');
```

Benchmark the target computer using the five predefined models, delete the build files, and return a structure array with results.

Start the target computer and run confidence test.

```
xpcbench_prework_snippet
```

Benchmark the target computer, delete the build files, and return a structure array.

```
current_results = xpcbench('this','-cleanup');  
current_results(1)
```

```
### Starting xPC Target build procedure for model:
    xpcminimal
### Successful completion of build procedure for model:
    xpcminimal
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: xpcminimal

### Starting xPC Target build procedure for model:
    f14tmp1
### Successful completion of build procedure for model:
    f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp1

### Starting xPC Target build procedure for model:
    f14tmp5
### Successful completion of build procedure for model:
    f14tmp5
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp5

### Starting xPC Target build procedure for model:
    f14tmp10
### Successful completion of build procedure for model:
    f14tmp10
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp10

### Starting xPC Target build procedure for model:
```

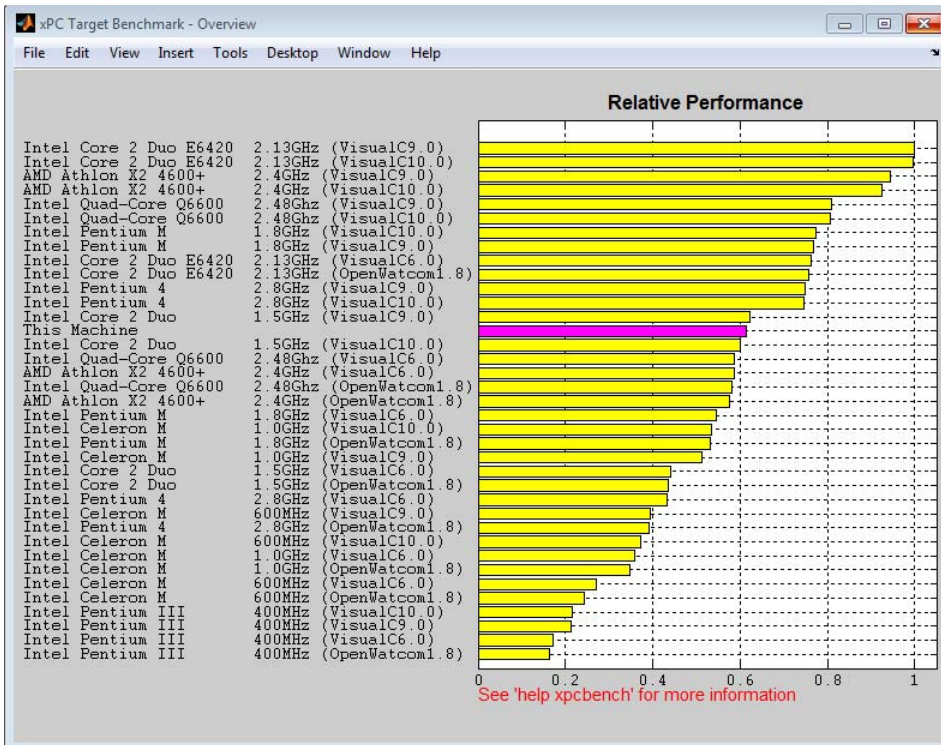
```
    f14tmp25
### Successful completion of build procedure for model:
    f14tmp25
### Looking for target: TargetPC1
### Download model onto target: TargetPC1

### Running benchmark for model: f14tmp25

ans =

    Name: 'Minimal'
    nBlocks: 3
    BuildTime: 23.8346
    BenchTime: 23.3861
    Tmin: 1.3328e-05
```





Minimal achievable sample times in  $\mu$ s

	Minimal	F14	F14*5	F14*10	F14*25	
Intel Core 2 Duo E6420	2.13GHz (VisualC9.0)	8	9	11	14	21
Intel Core 2 Duo E6420	2.13GHz (VisualC10.0)	8	10	11	13	22
AMD Athlon X2 4600+	2.4GHz (VisualC9.0)	8	9	11	14	25
AMD Athlon X2 4600+	2.4GHz (VisualC10.0)	8	9	12	14	25
Intel Quad-Core Q6600	2.48Ghz (VisualC9.0)	8	12	13	16	30
Intel Quad-Core Q6600	2.48Ghz (VisualC10.0)	8	10	13	17	30
Intel Pentium M	1.8GHz (VisualC10.0)	8	8	13	18	36
Intel Pentium M	1.8GHz (VisualC9.0)	8	10	13	17	36
Intel Core 2 Duo E6420	2.13GHz (VisualC6.0)	8	10	13	19	33
Intel Core 2 Duo E6420	2.13GHz (OpenWatcom1.8)	8	9	14	19	33
Intel Pentium 4	2.8GHz (VisualC9.0)	8	10	14	17	37
Intel Pentium 4	2.8GHz (VisualC10.0)	8	10	13	18	38
Intel Core 2 Duo	1.5GHz (VisualC9.0)	9	9	18	23	43
<b>This Machine</b>		<b>13</b>	<b>15</b>	<b>18</b>	<b>21</b>	<b>35</b>
Intel Core 2 Duo	1.5GHz (VisualC10.0)	9	12	17	24	44
Intel Quad-Core Q6600	2.48GHz (VisualC6.0)	8	13	18	24	47
AMD Athlon X2 4600+	2.4GHz (VisualC6.0)	8	11	16	25	55
Intel Quad-Core Q6600	2.48GHz (OpenWatcom1.8)	8	13	17	26	48
AMD Athlon X2 4600+	2.4GHz (OpenWatcom1.8)	8	11	16	26	58
Intel Pentium M	1.8GHz (VisualC6.0)	8	11	17	27	65
Intel Celeron M	1.0GHz (VisualC10.0)	9	9	19	27	66
Intel Pentium M	1.8GHz (OpenWatcom1.8)	8	10	18	29	67
Intel Celeron M	1.0GHz (VisualC9.0)	9	11	20	27	66
Intel Core 2 Duo	1.5GHz (VisualC6.0)	9	12	23	34	84
Intel Core 2 Duo	1.5GHz (OpenWatcom1.8)	8	13	24	36	83
Intel Pentium 4	2.8GHz (VisualC6.0)	8	12	22	40	90
Intel Celeron M	600MHz (VisualC9.0)	8	14	24	44	108
Intel Pentium 4	2.8GHz (OpenWatcom1.8)	9	13	24	44	101
Intel Celeron M	600MHz (VisualC10.0)	10	14	25	44	108
Intel Celeron M	1.0GHz (VisualC6.0)	8	14	27	46	153
Intel Celeron M	1.0GHz (OpenWatcom1.8)	10	13	28	50	123
Intel Celeron M	600MHz (VisualC6.0)	9	16	40	76	197
Intel Celeron M	600MHz (OpenWatcom1.8)	10	17	46	89	209
Intel Pentium III	400MHz (VisualC10.0)	12	17	50	96	259
Intel Pentium III	400MHz (VisualC9.0)	13	16	51	97	264
Intel Pentium III	400MHz (VisualC6.0)	12	22	70	146	378
Intel Pentium III	400MHz (OpenWatcom1.8)	11	22	84	172	405

**current\_results =**  
**xpcbench('f14', '-verbose', '-reboot', '-cleanup')**

Benchmark the target computer using the f14 model and all control options and return a structure array with results.

Start the target computer and run confidence test.

xpcbench\_prework\_snippet

Build 'f14', print build messages, run benchmark, delete build files, restart, and return results.

current\_results = xpcbench('f14', '-verbose', '-reboot',

```
'-cleanup')

### Starting xPC Target build procedure for model: f14tmp1
### Generating code into build folder: f14tmp1_xpc_rtw
### Invoking Target Language Compiler on f14tmp1.rtw
### Using System Target File: xpctarget.tlc
### Loading TLC function libraries
.
.
.
### Created DLM ..\f14tmp1.dlm
### Successful completion of build procedure for model:
      f14tmp1
### Looking for target: TargetPC1
### Download model onto target: TargetPC1
### Create xPC Object tg

xPC Object

      Connected          = Yes
      Application        = f14tmp1
      Mode               = Real-Time Single-Tasking
      Status             = stopped
      CPUOverload        = none
.
.
.
### Running benchmark for model: f14tmp1
### Reboot target: TargetPC1..... OK

Benchmark results for model:                F14*1
Number of blocks in model:                  62
Elapsed time for model build (sec):         15.4
Elapsed time for model benchmark (sec):     23.4
Minimal achievable sample time (microsec): 15.8

current_results =
```

# xpcbench

---

```
Name: 'F14*1'  
nBlocks: 62  
BuildTime: 15.3662  
BenchTime: 23.3516  
TsmIn: 1.5750e-05
```

**See Also** xpctest

## External Web Sites

- [http://www.mathworks.com/support/compilers/current\\_release/](http://www.mathworks.com/support/compilers/current_release/)

**Purpose** Create xPC Target boot disk or DOS Loader files and confirm current environment properties

**Syntax** `xpcbootdisk`

**Description** `xpcbootdisk` creates an xPC Target boot floppy, CD or DVD boot image, network boot image, or DOS Loader files for the current xPC Target environment. Use the `setxpcenv` function to set environment properties.

- Creating an xPC Target boot floppy consists of writing the bootable kernel image onto the disk. You are asked to insert an empty formatted floppy disk into the drive. At the end, a summary of the creation process is displayed.
- Creating an xPC Target CD/DVD boot image consists of creating the bootable kernel image in a designated area. You can then burn the files to a blank CD/DVD. If you have Microsoft® Windows Vista™ or Microsoft Windows® XP Service Pack 2 or 3 with Image Mastering API v2.0 (IMAPIv2.0), `xpcbootdisk` offers to create to the CD or DVD. Otherwise, you must use alternate third-party CD/DVD writing software to write ISO® image files.
- Creating an xPC Target network boot image consists of running `xpcnetboot` to start the network boot server process.
- Creating xPC Target DOS Loader files consists of creating the files in a designated area. You can then copy the files to the target computer flash disk.

If you update the environment, you need to update the target boot floppy, CD boot image, network boot image, or DOS Loader files for the new xPC Target environment with the function `xpcbootdisk`.

**Examples** To create a boot floppy disk, in the MATLAB window, type:

```
xpcbootdisk
```

**See Also** `setxpcenv` | `getxpcenv`

## **How To**

- “Target Boot Methods”
- “Command-Line Target Boot Methods”

**Purpose** Generate file suitable for use by From File block

**Syntax** `xpcbytes2file(filename,var1,...,varn)`

## Arguments

<code>filename</code>	Name of the data file from which the From File block distributes data.
<code>var1, . . . ,varn</code>	Column of data to be output to the model.

## Description

`xpcbytes2file(filename,var1,...,varn)` outputs one column of `var1, . . . ,varn` from file `filename` at every time step. All variables must have the same number of columns; the number of rows and data types can differ.

---

**Note** You might have the data organized such that a row refers to a single time step and not a column. In this case, pass to `xpcbytes2file` the transpose of the variable. To optimize file writes, organize the data in columns.

---

## Examples

In the following example, to use the From File block to output a variable `errorval` (single precision, scalar) and `velocity` (double, width 3) at every time step, you can generate the file with the command:

```
xpcbytes2file('myfile', errorval, velocity)
```

where `errorval` has class `'single'` and dimensions `[1 x N]` and `velocity` has class `'double'` and dimensions `[3 x N]`.

Set up the From File block to output

```
28 bytes  
(1 * sizeof('single') + 3 * sizeof('double'))
```

at every sample time.

# xpcexplr

---

<b>Purpose</b>	Open xPC Target Explorer
<b>Syntax</b>	xpcexplr
<b>Description</b>	<p>xpcexplr runs on the host computer and allows you to:</p> <ul style="list-style-type: none"><li>• Enter and change xPC Target environment properties</li><li>• Create an xPC Target bootable image</li><li>• Download, unload, and run target applications</li><li>• Monitor signals</li><li>• Tune parameters</li><li>• Add, remove, and configure xPC Target scopes</li><li>• Browse the target file system</li></ul>
<b>See Also</b>	setxpcenv   getxpcenv   xpcbootdisk
<b>How To</b>	<ul style="list-style-type: none"><li>• “Ethernet Communication Setup”</li><li>• “RS-232 Communication Setup”</li><li>• “Target Boot Methods”</li></ul>



<b>Purpose</b>	Compiler settings for xPC Target environment
<b>Syntax</b>	<pre>type = xpcgetCC type = xpcgetCC('Type') [type, location] = xpcgetCC location= xpcgetCC('Location') xpcgetCC('supported') xpcgetCC('installed') [compilers] = xpcgetCC('installed')</pre>
<b>Description</b>	<p><code>type = xpcgetCC</code> and <code>type = xpcgetCC('Type')</code> return the compiler type in <code>type</code>.</p> <p><code>[type, location] = xpcgetCC</code> returns the compiler type and its location in <code>type</code> and <code>location</code>.</p> <p><code>location= xpcgetCC('Location')</code> returns the compiler location in <code>location</code>.</p> <p><code>xpcgetCC('supported')</code> lists supported compiler versions for the xPC Target environment.</p> <p><code>xpcgetCC('installed')</code> lists the xPC Target supported compilers installed on the current host computer</p> <p><code>[compilers] = xpcgetCC('installed')</code> returns the xPC Target supported compilers installed on the current host computer in a structure.</p>
<b>Examples</b>	<p>Return the compiler type.</p> <pre>type = xpcgetCC</pre> <p>Return the compiler type and compiler location.</p> <pre>&gt;&gt; [type, location] = xpcgetCC</pre> <p>Return the xPC Target supported compilers installed on the current host computer in a structure and access the structure fields</p>

# xpcgetCC

---

```
[compilers] = xpcgetCC('installed')
```

```
compilers =
```

```
1x3 struct array with fields:
```

```
    Type
```

```
    Name
```

```
    Location
```

```
compilers.Type
```

```
ans =
```

```
VisualC
```

## See Also

```
xpcsetCC
```

**Purpose** Create kernel to boot target computer over dedicated network

**Syntax**  
`xpcnetboot`  
`xpcnetboot targetPCName`

**Arguments**      *targetPCName*      Target computer name as identified in xPC Target Explorer.

**Description** `xpcnetboot` creates an xPC Target kernel from which a target computer within the same network can start. This function also starts the following services as server processes:

- Bootstrap protocol (bootp) — `xpcbootpserver.exe`
- Trivial file transfer protocol (tftp) — `xpctftpserver.exe`

These processes respond to network boot requests from the target computer.

`xpcnetboot` without an argument creates a kernel for the default target computer (as identified in xPC Target Explorer).

`xpcnetboot targetPCName` creates an xPC Target kernel and waits for a request from the target computer named *targetPCName* (as identified in xPC Target Explorer).

**Examples** In the following example, `xpcnetboot` creates an xPC Target kernel and waits for a request from the target computer, TargetPC1.

```
xpcnetboot TargetPC1
```

# xpcsetCC

---

**Purpose** Compiler settings for xPC Target environment

**Syntax**

```
xpcsetCC('setup')
xpcsetCC('location')
xpcsetCC('type')
xpcsetCC(type, location)
```

**Description** `xpcsetCC('setup')` queries the host computer for installed C compilers that the xPC Target environment supports. You can then select the C compiler.

`xpcsetCC('location')` sets the compiler location.

`xpcsetCC('type')` sets the compiler type. '*type*' must be VISUALC, representing the Microsoft Visual Studio® C compiler.

`xpcsetCC(type, location)` sets the compiler type and location.

**See Also** `xpcgetCC`

**How To**

- “Command-Line C Compiler Configuration”

**Purpose** Package for xPC Target MATLAB classes

**Description** Use xpctarget package objects to access the MATLAB command line capabilities.

### Functions

Assign these object creation functions to a MATLAB variable to get access to the properties and methods of the class.

Function	Description
xpctarget.fs	Create file system object
xpctarget.ftp	Create file transfer protocol (FTP) object
xpctarget.targets	Create container object to manage target computer environment collection objects
xpctarget.xpc	Create target object representing target application

# xpctarget.env Class

---

**Purpose** Stores target environment properties

**Description** Each `xpctarget.env` Class object contains the environment properties for a particular target computer. A collection of these objects is stored in an `xpctarget.targets` Class object. An individual object in a collection is accessed via the `xpctarget.targets.Item (env collection object)` method.

## Methods


Method	Description
<code>xpctarget.env.get (env object)</code>	Return property values for an environment object
<code>xpctarget.env.set (env object)</code>	Change property values for an environment object

## Properties

The environment properties define communication between the host computer and target computer and the type of target boot floppy created during the setup process. An understanding of the environment properties will help you configure the xPC Target environment.

---

**Tip** To access a subset of these properties in xPC Target Explorer:

- 1 Expand a target computer node in the **Targets** pane.
- 2 Click the Target Properties icon  in the toolbar or double-click **Properties**.

- 
- Host-to-Target Communication on page 71
  - Target Settings on page 77
  - Boot Configuration on page 81

### Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.</p>

# xpctarget.env Class

Environment Property	Description
	<hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>



Environment Property	Description
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to 255.255.255.255, which means that a gateway is not used to connect to the target computer.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>

# xpctarget.env Class

---

Environment Property	Description
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>

Environment Property	Description
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings</p>

# xpctarget.env Class

---

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
TcpIpTargetISAMemPort	<p>Property value is '0xnxxx'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the</p>

Environment Property	Description
	<p><b>Target Properties</b> pane of xPC Target Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer.</p>

### Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.</p>

## xpctarget.env Class

Environment Property	Description
MaxModelSize	<p>Property values are '1MB' (the default), '4MB', and '16MB'.</p> <p>Select 1 MB, 4 MB, or 16 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.</p> <ul style="list-style-type: none"><li>• BootFloppy and DOSLoader modes ignore this value.</li><li>• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.</li></ul>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.</p>
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

## xpctarget.env Class

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of xPC Target Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.</p> <p>The xPC Target kernel can use only 2 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p>



Environment Property	Description
	Set this value to 'on' if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to 'off'.

## Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are 'Disabled' and 'Enabled'. This property is read only.  Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

# xpctarget.env Class

Environment Property	Description
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of xPC Target Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p> <hr/>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p>

Environment Property	Description
	<p data-bbox="914 369 1166 392">xx:xx:xx:xx:xx:xx</p> <p data-bbox="914 435 1322 847">To update the MAC address in xPC Target Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.</p>

# xpctarget.env.get (env object)

---

**Purpose** Return target environment property values

**Syntax**

```
property_value = env_object.property_name  
property_value = env_object.get('property_name')  
property_value = get(env_object, 'property_name')  
property_value = env_object.get  
property_value = get(env_object)
```

**Arguments**

env_object	Name of a target environment object.
property_name	Name of a target environment object property.

**Description** property\_value = env\_object.property\_name gets the current value of property property\_name from target environment object env\_object. Alternative syntaxes are:

```
property_value = env_object.get('property_name')
```

```
property_value = get(env_object, 'property_name')
```

property\_value = env\_object.get gets the values of all properties of target environment object env\_object. An alternative syntax is:

```
property_value = get(env_object)
```


Get an individual environment object with the xpctarget.targets.Item (env collection object) method. For example:

```
tgs=xpctarget.targets;  
env_object=tgs.Item('TargetPC1');  
property_value=env_object.HostTargetComm
```

The environment properties for a target environment object are listed in the following tables.

---

**Tip** To access a subset of these properties in xPC Target Explorer:

- 1 Expand a target computer node in the **Targets** pane.
- 2 Click the Target Properties icon  in the toolbar or double-click **Properties**.

- 
- “Host-to-Target Communication” on page 1-85
  - “Target Settings” on page 1-91
  - “Boot Configuration” on page 1-95

## Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.</p>

## xpctarget.env.get (env object)

Environment Property	Description
	<hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>

Environment Property	Description
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to 255.255.255.255, which means that a gateway is not used to connect to the target computer.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>

## xpctarget.env.get (env object)

---

Environment Property	Description
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>



Environment Property	Description
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings</p>

## xpctarget.env.get (env object)

---

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
TcpIpTargetISAMemPort	<p>Property value is '0xnxxx'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the</p>

Environment Property	Description
	<p><b>Target Properties</b> pane of xPC Target Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer.</p>

## Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <math>n</math> indicates the index number for the Ethernet card on a target computer. Note that the <math>(n-1)</math>th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.</p>

## xpctarget.env.get (env object)

Environment Property	Description
MaxModelSize	<p>Property values are '1MB' (the default), '4MB', and '16MB'.</p> <p>Select 1 MB, 4 MB, or 16 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.</p> <ul style="list-style-type: none"><li>• BootFloppy and DOSLoader modes ignore this value.</li><li>• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.</li></ul>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.</p>
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

## xpctarget.env.get (env object)

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of xPC Target Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.</p> <p>The xPC Target kernel can use only 2 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p>

Environment Property	Description
	Set this value to 'on' if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to 'off'.

## Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	Property values are 'Disabled' and 'Enabled'. This property is read only.  Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.

## xpctarget.env.get (env object)

Environment Property	Description
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of xPC Target Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p> <hr/>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p>



Environment Property	Description
	<p>xx:xx:xx:xx:xx:xx</p> <p>To update the MAC address in xPC Target Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.</p>

## See Also

xpctarget.env.set (env object)

# xpctarget.env.set (env object)

---

**Purpose** Change target environment object property values

**Syntax**

```
env_object.property_name = property_value  
env_object.set('prop_name1','prop_value1','prop_name2',. . .)  
set(env_object,'prop_name1','prop_value1','prop_name2',. . .)
```

**Arguments**

env_object	Name of a target environment object.
property_name	Name of a target environment object property.
property_value	Value for a target environment object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

**Description**

env\_object.property\_name = property\_value sets property property\_name of target environment object env\_object to property\_value. Alternative syntaxes for one or more property-value pairs are:

```
env_object.set('prop_name1','prop_value1','prop_name2',. . .  
. )  
set(env_object,'prop_name1','prop_value1','prop_name2',. . .  
. )
```


Get an individual environment object with the xpctarget.targets.Item (env collection object) method. For example:

```
tgs=xpctarget.targets;  
env_object=tgs.Item('TargetPC1');  
env_object.HostTargetComm='RS232'
```

Not all properties are user writable. The writable properties for a target environment object are listed in the following table.

---

**Tip** To access a subset of these properties in xPC Target Explorer:

- 1 Expand a target computer node in the **Targets** pane.
- 2 Click the Target Properties icon  in the toolbar or double-click **Properties**.

- 
- “Host-to-Target Communication” on page 1-99
  - “Target Settings” on page 1-105
  - “Boot Configuration” on page 1-109

## Host-to-Target Communication

Environment Property	Description
HostTargetComm	<p>Property values are 'RS232' and 'TcpIp'.</p> <p>Select RS-232 or TCP/IP from the <b>Communication type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you select RS-232, you also must set the property RS232HostPort. If you select TCP/IP, then you also need to set all properties that start with TcpIp.</p>

## xpctarget.env.set (env object)

Environment Property	Description
	<hr/> <p><b>Note</b> RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.</p> <hr/>
RS232Baudrate	<p>Property values are '115200', '57600', '38400', '19200', '9600', '4800', '2400', and '1200'.</p> <p>Select 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200 from the <b>Baud rate</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
RS232HostPort	<p>Property values are 'COM1' and 'COM2'.</p> <p>Select COM1 or COM2 from the <b>Host port</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. The software automatically determines the COM port on the target computer.</p> <p>Before you can select an RS-232 port, you need to set the HostTargetComm property to RS232.</p>

Environment Property	Description
TcpIpGateway	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the IP address for your gateway in the <b>Gateway</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to 255.255.255.255, which means that a gateway is not used to connect to the target computer.</p> <p>If you communicate with your target computer from within a LAN that uses gateways, and your host and target computers are connected through a gateway, you must enter a value for this property. If your LAN does not use gateways, you do not need to change this property. Ask your system administrator.</p>
TcpIpSubNetMask	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter the subnet mask of your LAN in the <b>Subnet mask</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, your subnet mask could be 255.255.255.0.</p>

## xpctarget.env.set (env object)

---

Environment Property	Description
TcpIpTargetAddress	<p>Property value is 'xxx.xxx.xxx.xxx'.</p> <p>Enter a valid IP address for your target computer in the <b>IP address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer. Ask your system administrator for this value.</p> <p>For example, 192.168.0.10.</p>
TcpIpTargetBusType	<p>Property values are 'PCI', 'ISA', and 'USB'.</p> <p>Select PCI, ISA, or USB from the <b>Bus type</b> list in the <b>Target Properties</b> pane of xPC Target Explorer. This property is set by default to PCI, and determines the bus type of your target computer. You do not need to define a bus type for your host computer, which can be the same or different from the bus type in your target computer.</p> <p>If TcpIpTargetBusType is set to PCI, then the properties TcpIpISAMemPort and TcpIpISAIRQ have no effect on TCP/IP communication.</p> <p>If you are using an ISA bus card, set TcpIpTargetBusType to ISA and enter values for TcpIpISAMemPort and TcpIpISAIRQ.</p>

Environment Property	Description
TcpIpTargetDriver	<p>Property values are '3C90x', 'I8254x', 'I82559', 'NE2000', 'NS83815', 'R8139', 'R8168', 'Rhine', 'RTLANCE', 'SMC91C9X', 'USBAX772', 'USBAX172', and 'Auto'.</p> <p>Select THREECOM_3C90x, INTEL_I8254x, INTEL_I82559, NE2000, NS83815, R8139, R8168, Rhine, RTLANCE, SMC91C9X, USBAX772, USBAX172, or Auto from the <b>Target driver</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p>
TcpIpTargetISAIRQ	<p>Property value is 'n', where <i>n</i> is between 5 and 15 inclusive.</p> <p>Select an IRQ value from the <b>IRQ</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on the ISA-bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O-port base address by moving the jumpers on the card.</p> <p>Set the IRQ to 5, 10, or 11. If one of these hardware settings</p>

## xpctarget.env.set (env object)

---

Environment Property	Description
	leads to a conflict in your target computer, choose another IRQ and make the corresponding changes to your jumper settings.
TcpIpTargetISAMemPort	<p>Property value is '0xnxxx'.</p> <p>Enter an I/O port base address in the <b>Address</b> box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you are using an ISA bus Ethernet card, you must enter values for the properties TcpIpISAMemPort and TcpIpISAIRQ. The values of these properties must correspond to the jumper settings or ROM settings on your ISA bus Ethernet card.</p> <p>On your ISA bus card, assign an IRQ and I/O port base address by moving the jumpers on the card.</p> <p>Set the I/O port base address to around 0x300. If one of these hardware settings leads to a conflict in your target computer, choose another I/O port base address and make the corresponding changes to your jumper settings.</p>
TcpIpTargetPort	<p>Property value is 'xxxxx'.</p> <p>Enter a port address greater than 20000 in the <b>Port</b> box in the</p>



Environment Property	Description
	<p><b>Target Properties</b> pane of xPC Target Explorer.</p> <p>This property is set by default to 22222. The default value is higher than the reserved area (telnet, ftp, . . .) and is only of use on the target computer.</p>

## Target Settings

Environment Property	Description
EthernetIndex	<p>Property value is 'n', where <i>n</i> indicates the index number for the Ethernet card on a target computer. Note that the (n-1)th Ethernet card on the target computer has an index number 'n'. The default index number is 0.</p> <p>If the target computer has multiple Ethernet cards, you must select one of the cards for host-target communication. This option returns the index number of the card selected on the target computer upon booting.</p>
LegacyMultiCoreConfig	<p>Property values are 'on' and 'off' (the default).</p> <p>Set this value to 'on' only if your target computer contains hardware not compliant with the Advanced Configuration and Power Interface (ACPI) standard. Otherwise, set this value to 'off'.</p>

## xpctarget.env.set (env object)

Environment Property	Description
MaxModelSize	<p>Property values are '1MB' (the default), '4MB', and '16MB'.</p> <p>Select 1 MB, 4 MB, or 16 MB from the <b>Model size</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Choosing the maximum model size reserves the specified amount of memory on the target computer for the target application. Memory not used by the target application is used by the kernel and by the heap for data logging.</p> <p>Selecting too high a value leaves less memory for data logging. Selecting too low a value does not reserve enough memory for the target application and creates an error. You can approximate the size of the target application by the size of the DLM file produced by the build process.</p> <ul style="list-style-type: none"><li>• BootFloppy and DOSLoader modes ignore this value.</li><li>• In StandAlone mode, you can only use MaxModelSize values '1MB' and '4MB'.</li></ul>
MulticoreSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Multicore CPU</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your target computer has multicore processors, set this value to 'on' to take advantage of these processors for background tasks. Otherwise, set this value to 'off'.</p>
Name	Target computer name.

Environment Property	Description
NonPentiumSupport	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Target is a 386/486</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' if your target computer has a 386 or 486 compatible processor. Otherwise, set it to 'off'. If your target computer has a Pentium or higher compatible processor, selecting this check box slows the performance of your target computer.</p>
SecondaryIDE	<p>Property values are 'on' and 'off' (the default).</p> <p>Select or clear the <b>Secondary IDE</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>Set this value to 'on' only if you want to use the disks connected to a secondary IDE controller. If you do not have disks connected to the secondary IDE controller, leave this value set to 'off'.</p>
ShowHardware	<p>Property values are 'on' and 'off' (the default).</p> <p>If you create a target boot kernel when ShowHardware is 'on' and boot the target computer with it, the kernel displays the index, bus, slot, function, and target driver for each Ethernet card on the target monitor.</p> <p>The host computer cannot communicate with the target computer after the kernel boots with ShowHardware set.</p>

## xpctarget.env.set (env object)

Environment Property	Description
TargetRAMSizeMB	<p>Property values are 'Auto' (the default) and 'xxx', where xxx is a positive value specifying the amount of RAM, in megabytes, installed on the target computer.</p> <p>Under <b>RAM size</b>, click the <b>Auto</b> or <b>Manual</b> button in the <b>Target Properties</b> pane of xPC Target Explorer. If you click <b>Manual</b>, enter the amount of RAM, in megabytes, installed on the target computer in the <b>Size(MB)</b> box.</p> <p>TargetRAMSizeMB defines the total amount of installed RAM in the target computer. This RAM is used for the kernel, target application, data logging, and other functions that use the heap.</p> <p>If TargetRAMSizeMB is assigned 'Auto', the target application reads the target computer BIOS and determines the amount of memory up to a maximum of 2 GB. If the target application cannot read the BIOS, you must select Manual mode and enter the amount of memory, in megabytes, up to a maximum of 2 GB.</p> <p>The xPC Target kernel can use only 2 GB of memory.</p>
TargetScope	<p>Property values are 'Disabled' and 'Enabled' (the default).</p> <p>Select or clear the <b>Graphics mode</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If you set TargetScope to Disabled, the target computer displays information as text.</p> <p>To use the full features of a target scope, install a keyboard on the target computer.</p>
USBSupport	<p>Property values are 'on' (the default) and 'off'.</p> <p>Select or clear the <b>USB Support</b> check box in the <b>Target Properties</b> pane of xPC Target Explorer.</p>

Environment Property	Description
	Set this value to 'on' if you want to use a USB port on the target computer; for example, to connect a USB mouse. Otherwise, set it to 'off'.

## Boot Configuration

Environment Property	Description
BootFloppyLocation	Drive name for creation of target boot disk.
CDBootImageLocation	Location of cdboot.iso file for creation of CD target boot disk.
DOSLoaderLocation	Location of DOSLoader files to boot target computers from devices other than floppy disk or CD.
EmbeddedOption	<p>Property values are 'Disabled' and 'Enabled'. This property is read only.</p> <p>Note that the xPC Target Embedded Option product is enabled only if you purchase an additional license.</p>

## xpctarget.env.set (env object)

Environment Property	Description
TargetBoot	<p>Property values are 'BootFloppy', 'CDBoot', 'DOSLoader', 'NetworkBoot', and 'StandAlone'.</p> <p>Select Removable Disk, CD, DOS Loader, Network, or Stand Alone from the <b>Boot mode</b> list in the <b>Target Properties</b> pane of xPC Target Explorer.</p> <p>If your license file does not include the license for the xPC Target Embedded Option product, your only options are Removable Disk, CD, DOS Loader, and Network. With the xPC Target Embedded Option product licensed and installed, you have the additional choice of Stand Alone.</p> <hr/> <p><b>Tip</b> In the <b>Target Properties</b> pane of xPC Target Explorer, click the <b>Create boot disk</b> button to create a bootable image in the specified boot mode.</p> <hr/>
TargetMACAddress	<p>Physical target computer MAC address from which to accept boot requests when booting within a dedicated network. Format the MAC address as six pairs of hexadecimal numbers, separated by colons:</p>

Environment Property	Description
	<p>xx:xx:xx:xx:xx:xx</p> <p>To update the MAC address in xPC Target Explorer, first click the <b>Reset</b> button in the <b>Target Properties</b> pane. You can then click the <b>Specify new MAC address</b> button to enter a MAC address manually in the <b>MAC address</b> box. If you do not enter a MAC address manually, the software will obtain the MAC address automatically the next time you restart the target computer.</p>

### See Also

xpctarget.env.get (env object)

# xpctarget.fs Class

---

**Purpose** Manage the folders and files on the target computer

**Description** This class includes the folder methods from `xpctarget.fsbase Class` and implements file access methods used on the target computer.

## Constructor

Constructor	Description
<code>xpctarget.fs</code>	Create file system object

## Methods

These methods are inherited from `xpctarget.fsbase Class`.

Method	Description
<code>xpctarget.fsbase.cd</code>	Change folder on target computer
<code>xpctarget.fsbase.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbase.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbase.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbase.rmdir</code>	Remove folder from target computer

These methods are specific to class `fs`.

Method	Description
<code>xpctarget.fs.diskinfo</code>	Information about target computer drive
<code>xpctarget.fs.fclose</code>	Close open target computer file(s)
<code>xpctarget.fs.fileinfo</code>	Target computer file information
<code>xpctarget.fs.filetable</code>	Information about open files in target computer file system



<b>Method</b>	<b>Description</b>
<code>xpctarget.fs.fopen</code>	Open target computer file for reading
<code>xpctarget.fs.fread</code>	Read open target computer file
<code>xpctarget.fs.fwrite</code>	Write binary data to open target computer file
<code>xpctarget.fs.getfilesize</code>	Size of file on target computer
<code>xpctarget.fs.removefile</code>	Remove file from target computer

# xpctarget.fs

---

**Purpose** Create xPC Target file system object

**Syntax**  
fileSYS\_object = xpctarget.fs  
fileSYS\_object = xpctarget.fs('mode', 'arg1', 'arg2')

**Arguments**

fileSYS_object	Variable name to reference the file system object.
mode	Optionally, enter the communication mode:

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

	TCPIP	Specify TCP/IP connection with target computer.
	RS232	Specify RS-232 connection with target computer.
arg1		Optionally, enter an argument based on the mode value:
	IP address	If mode is 'TCPIP', enter the IP address of the target computer.
	COM port	If mode is 'RS232', enter the host COM port.
arg2		Optionally, enter an argument based on the mode value:
	Port	If mode is 'TCPIP', enter the port number for the target computer.
	Baud rate	If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

## Description

Constructor of a file system object (`xpctarget.fs` Class). The file system object represents the file system on the target computer. You work with the file system by changing the file system object using methods.

`filesystem_object = xpctarget.fs` constructs a file system object for the case where you have one target computer object or designate a target computer as the default one in your system.

`filesystem_object = xpctarget.fs('mode', 'arg1', 'arg2')` constructs a file system object for the case where you have multiple target computers in your system or want to identify a target computer with the file system object.

## Examples

In the following example, a file system object for a target computer with an RS-232 connection is created.

```
fs1=xpctarget.fs('RS232', 'COM1', '115200')
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.fs` object by passing the `xpctarget.xpc` object variable to the `xpctarget.fs` constructor as an argument.

```
tg1=xpctarget.xpc('RS232', 'COM1', '115200');  
fs2=xpctarget.fs(tg1)
```

# xpctarget.fs.diskinfo

---

**Purpose** Information about target computer drive

**Syntax** `filesys_obj.diskinfo(target_PC_drive)`  
`diskinfo(filesys_obj,target_PC_drive)`

**Arguments**

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>target_PC_drive</code>	Name of the target computer drive for which to return information.

**Description** `filesys_obj.diskinfo(target_PC_drive)` returns disk information for the specified target computer drive. An alternative syntax is:  
`diskinfo(filesys_obj,target_PC_drive)`  
This is a method of `xpctarget.fs` objects called from the host computer.

## Examples

Return disk information for the target computer C:\ drive for the file system object fsys.

```
diskinfo(fsys,'C:\') or fsys.diskinfo('C:\')  
ans =
```

```
          Label: 'SYSTEM '  
    DriveLetter: 'C'  
      Reserved: ''  
  SerialNumber: 1.0294e+009  
FirstPhysicalSector: 63  
          FATType: 32  
        FATCount: 2  
    MaxDirEntries: 0  
    BytesPerSector: 512  
SectorsPerCluster: 4  
    TotalClusters: 2040293  
    BadClusters: 0  
    FreeClusters: 1007937  
          Files: 19968  
    FileChains: 22480  
    FreeChains: 1300  
LargestFreeChain: 64349
```

# xpctarget.fs.fclose

---

**Purpose** Close open target computer file(s)

**Syntax** `fclose(filesys_obj, file_ID)`  
`filesys_obj.fclose(file_ID)`

**Arguments**

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>file_ID</code>	File identifier of the file to close.

**Description** Method of `xpctarget.fs` objects. From the host computer, closes one or more open files in the target computer file system (except standard input, output, and error). The `file_ID` argument is the file identifier associated with an open file (see `xpctarget.fs.fopen` and `xpctarget.fs.filetable`). You cannot have more than eight files open in the file system.

**Examples** Close the open file identified by the file identifier `h` in the file system object `fsys`.

```
fclose(fsys,h) or fsys.fclose(h)
```

**See Also** `fclose` | `xpctarget.fs.fopen` | `xpctarget.fs.fread` | `xpctarget.fs.filetable` | `xpctarget.fs.fwrite`

**Purpose** Target computer file information

**Syntax** `fileinfo(filesys_obj,file_ID)`  
`filesys_obj.fileinfo(file_ID)`

**Arguments**

<code>filesys_obj</code>	Name of the <code>xpctarget.fs</code> file system object.
<code>file_ID</code>	File identifier of the file for which to get file information.

**Description** Method of `xpctarget.fs` objects. From the host computer, gets the information for the file associated with `file_ID`.

**Examples** Return file information for the file associated with the file identifier `h` in the file system object `fsys`.

```
fileinfo(fsys,h) or fsys.fileinfo(h)
ans =
        FilePos: 0
        AllocatedSize: 12288
        ClusterChains: 1
        VolumeSerialNumber: 1.0450e+009
        FullName: 'C:\DATA.DAT'
```

# xpctarget.fs.filetable

---

**Purpose** Information about open files in target computer file system

**Syntax** filetable(filesys\_obj)  
filesys\_obj.filetable

**Arguments** filesys\_obj Name of the xpctarget.fs file system object.

**Description** Method of xpctarget.fs objects. From the host computer, displays a table of the open files in the target computer file system. You cannot have more than eight files open in the file system.

**Examples** Return a table of the open files in the target computer file system for the file system object fsys.

```
filetable(fsys) or fsys.filetable
ans =
Index      Handle  Flags      FilePos  Name
-----
      0  00060000  R__         8512  C:\DATA.DAT
      1  00080001  R__           0  C:\DATA1.DAT
      2  000A0002  R__         8512  C:\DATA2.DAT
      3  000C0003  R__         8512  C:\DATA3.DAT
      4  001E000S  R__           0  C:\DATA4.DAT
```

The table returns the open file handles in hexadecimal. To convert a handle to one that other xpctarget.fs methods, such as fclose, can use, use the hex2dec function.

```
h1 = hex2dec('001E0001')
h1 =
1966081
```

To close that file, use the xpctarget.fs fclose method.

```
fsys.fclose(h1);
```



**See Also**      [xpctarget.fs.fopen](#) | [xpctarget.fs.fclose](#)

# xpctarget.fs.fopen

---

**Purpose** Open target computer file for reading

**Syntax**

```
file_ID = fopen(file_obj, 'file_name')
file_ID = file_obj.fopen('file_name')
file_ID = fopen(file_obj, 'file_name', permission)
file_ID = file_obj.fopen('file_name', permission)
```

**Arguments**

file_obj	Name of the xpctarget.fs object.
'file_name'	Name of the target computer to open.
permission	Values are 'r', 'w', 'a', 'r+', 'w+', or 'a+'. This argument is optional with 'r' as the default value.

**Description** Method of xpctarget.fs objects. From the host computer, opens the specified filename on the target computer for binary access.

The permission argument values are

- 'r'  
Open the file for reading (default). The method does nothing if the file does not already exist.
- 'w'  
Open the file for writing. The method creates the file if it does not already exist.
- 'a'  
Open the file for appending to the file. Initially, the file pointer is at the end of the file. The method creates the file if it does not already exist.
- 'r+'  
Open the file for reading and writing. Initially, the file pointer is at the beginning of the file. The method does nothing if the file does not already exist.

- 'w+'

Open the file for reading and writing. The method empties the file first, if the file already exists and has content, and places the file pointer at the beginning of the file. The method creates the file if it does not already exist.

- 'a+'

Open the file for reading and appending to the file. Initially, the file pointer is at the beginning of the file. The method creates the file if it does not already exist.

You cannot have more than eight files open in the file system. This method returns the file identifier for the open file in `file_ID`. You use `file_ID` as the first argument to the other file I/O methods (such as `xpctarget.fs.fclose`, `xpctarget.fs.fread`, and `xpctarget.fs.fwrite`).

## Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h = fopen(fsys,'data.dat') or fsys.fopen('data.dat')
ans =
    2883584
d = fread(h);
```

## See Also

`fopen` | `xpctarget.fs.fclose` | `xpctarget.fs.fread` | `xpctarget.fs.fwrite`

# xpctarget.fs.fread

---

**Purpose** Read open target computer file

**Syntax**

```
A = file_obj.fread(file_ID)
A = fread(file_obj,file_ID)
A = file_obj.fread(file_ID,offset,numbytes)
A = fread(file_obj,file_ID,offset,numbytes)
```

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.
<code>file_ID</code>	File identifier of the file to read.
<code>offset</code>	Position from the beginning of the file from which <code>fread</code> can start to read.
<code>numbytes</code>	Maximum number of bytes <code>fread</code> can read.

**Description** `A = file_obj.fread(file_ID)` reads binary data from the file on the target computer and writes it into matrix `A`. The `file_ID` argument is the file identifier associated with an open file (see `xpctarget.fs.fopen`). An alternative syntax is:

```
A = fread(file_obj,file_ID)
```

`A = file_obj.fread(file_ID,offset,numbytes)` reads a block of bytes from `file_ID` and writes the block into matrix `A`. An alternative syntax is:

```
A = fread(file_obj,file_ID,offset,numbytes)
```

The `offset` argument specifies the position from the beginning of the file from which this function can start to read. `numbytes` specifies the maximum number of bytes to read.

To get a count of the total number of bytes read into `A`, use the following:

```
count = length(A);
```

`length(A)` might be less than the number of bytes requested if that number of bytes are not currently available. It is zero if the operation reaches the end of the file.

This is a method of `xpctarget.fs` objects called from the host computer.

## Examples

Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for reading.

```
h=fsys.fopen('data.dat')  
d=fread(h);
```

This reads the file `data.dat` and stores the contents of the file to `d`. This content is in the xPC Target file format.

## See Also

`fread` | `xpctarget.fs.fclose` | `xpctarget.fs.fopen` | `xpctarget.fs.fwrite`

# xpctarget.fs.fwrite

---

**Purpose** Write binary data to open target computer file

**Syntax** `fwrite(file_obj,file_ID,A)`  
`file_obj.fwrite(file_ID,A)`

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.
<code>file_ID</code>	File identifier of the file to write.
<code>A</code>	Elements of matrix <code>A</code> to be written to the specified file.

**Description** Method of `xpctarget.fs` objects. From the host computer, writes the elements of matrix `A` to the file identified by `file_ID`. The data is written to the file in column order. The `file_ID` argument is the file identifier associated with an open file (see `xpctarget.fs.fopen`). `fwrite` requires that the file be open with write permission.

**Examples** Open the file `data.dat` in the target computer file system object `fsys`. Assign the resulting file handle to a variable for writing.

```
h = fopen(fsys,'data.dat','w')
```

or

```
fsys.fopen('data.dat','w')
```

```
ans =  
    2883584  
d = fwrite(fsys,h,magic(5));
```

This writes the elements of matrix `A` to the file handle `h`. This content is written in column order.

**See Also** `fwrite` | `xpctarget.fs.fclose` | `xpctarget.fs.fopen` | `xpctarget.fs.fread`

**Purpose** Size of file on target computer

**Syntax** `getfilesize(file_obj,file_ID)`  
`file_obj.getfilesize(file_ID)`

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.
<code>file_ID</code>	File identifier of the file to get the size of.

**Description** Method of `xpctarget.fs` objects. From the host computer, gets the size (in bytes) of the file identified by the `file_ID` file identifier on the target computer file system. Use the xPC Target file object method `xpctarget.fs.fopen` to open the file system object.

**Examples** Get the size of the file identifier `h` for the file system object `fsys`.

```
getfilesize(fsys,h) or fsys.getfilesize(h)
```

**See Also** `xpctarget.fs.fopen`

# xpctarget.fs.readxpcfile

---

**Purpose** Interpret raw data from xPC Target file format

**Syntax** `readxpcfile('filename')`  
`file = readxpcfile(data)`

**Arguments**

<code>data_vector</code>	Vector of uint8 bytes.
<code>'filename'</code>	File from which the vector of uint8 bytes is read. Vector is written

**Description**

`readxpcfile('filename')` converts the contents of 'filename' to double precision data representing the signals and timestamps.

`file = readxpcfile(data)` converts vector `data_vector` to double precision data representing the signals and timestamps.

`readxpcfile` converts xPC Target file format content (in bytes) to double precision data. It returns a structure that contains the following fields:

- `version`  
Not used.
- `sector`  
Not used.
- `headersize`  
Not used.
- `numSignals`  
Array of signal names.
- `data`  
Array of signal data.
- `signalNames`



Cell array of signal names.

A file scope creates the data. After you download the data from a target computer, use one of the following to read the data:

- To read the data after you download it to the target computer, use the `fread` function
- To download the data to the target computer and read it, use the `xpctarget.fs.fread` method.

## Examples

Use the `xpctarget.fs` object to convert data:

```
file=xpctarget.fs;  
h=file.fopen('filename');  
data=file.fread(h);  
file.fclose(h);  
file = readxpcfile(data);
```

Use the `xpctarget.ftp` object to copy the file from the target computer to the host computer, then read and convert the data.

```
xpcftp=xpctarget.ftp  
xpcftp.get('filename')  
datafile = readxpcfile('filename') % Convert the data
```

Use the `xpctarget.ftp` object to copy the file from the target computer to the host computer, then read and convert the data separately.

```
xpcftp=xpctarget.ftp  
xpcftp.get('filename')  
handle=fopen('filename')  
data=fread(handle,'*uint8'); % Data should be read in uint8 format  
fclose(handle);  
data=data';  
datafile = readxpcfile(data); % Convert the data
```

## See Also

`xpctarget.ftp.get` (ftp) | `xpctarget.fs.fopen` | `xpctarget.fs.fread`

# xpctarget.fs.removefile

---

**Purpose** Remove file from target computer

**Syntax** `removefile(file_obj,file_name)`  
`file_obj.removefile(file_name)`

**Arguments**

<code>file_name</code>	Name of the file to remove from the target computer file system.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.

**Description** Method of `xpctarget.fs` objects. Removes a file from the target computer file system.

---

**Note** You cannot recover this file once it is removed.

---

**Examples** Remove the file `data2.dat` from the target computer file system `fsys`.

```
removefile(fsys, 'data2.dat')
```

or

```
fsys.removefile('data2.dat')
```

**Purpose** Select target computer drive

**Syntax** `selectdrive(file_obj, 'drive')`  
`file_obj.selectdrive('drive')`

**Arguments**

<code>drive</code>	Name of the drive to set.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.

**Description** Method of `xpctarget.fs` objects. `selectdrive` sets the current drive of the target computer to the specified string. Enter the drive string with an extra backslash (`\`). For example, `D:\` for the `D:\` drive.

---

**Note** Use the `xpctarget.fsbases.cd` method instead to get the same behavior.

---

**Examples** Set the current target computer drive to `D:\`.

```
selectdrive(fsys, 'D:\\')
```

or

```
fsys.selectdrive('D:\\')
```

# xpctarget.fsbase Class

---

**Purpose** Base class of file system and file transfer protocol (FTP) classes

**Description** This class is the base class for `xpctarget.fs` Class and `xpctarget.ftp` Class. All methods are inherited by the derived classes. The constructor for this class is called implicitly when the constructors for the derived classes are called:

## Methods

These methods are inherited by the derived classes.

Method	Description
<code>xpctarget.fsbase.cd</code>	Change folder on target computer
<code>xpctarget.fsbase.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbase.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbase.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbase.rmdir</code>	Remove folder from target computer

**Purpose** Change folder on target computer

**Syntax** `cd(file_obj,target_PC_dir)`  
`file_obj.cd(target_PC_dir)`

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<code>target_PC_dir</code>	Name of the target computer folder to change to.

**Description** Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, changes folder on the target computer.

**Examples** Change folder from the current to one named `logs` for the file system object `fsys`.

```
cd(fsys,logs) or fsys.cd(logs)
```

Change folder from the current to one named `logs` for the FTP object `f`.

```
cd(f,logs) or f.cd(logs)
```

**See Also** `cd` | `xpctarget.fsbase.mkdir` | `xpctarget.fsbase.pwd`

# xpctarget.fsbase.dir

---

**Purpose** List contents of current folder on target computer

**Syntax** `dir(file_obj)`

**Arguments** `file_obj` Name of the `xpctarget.ftp` or `xpctarget.fs` object.

**Description** Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. From the host computer, lists the contents of the current folder on the target computer.

To get the results in an M-by-1 structure, use a syntax like `ans=dir(file_obj)`. This syntax returns a structure like the following:

```
ans =  
1x5 struct array with fields:  
name  
date  
time  
bytes  
isdir
```

where

- `name` — Name of an object in the folder, shown as a cell array. The name, stored in the first element of the cell array, can have up to eight characters. The three-character file extension is stored in the second element of the cell array.
- `date` — Date of the last save of that object
- `time` — Time of the last save of that object
- `bytes` — Size in bytes of that object
- `isdir` — Logical value indicating that the object is (1) or is not (0) a folder

## Examples

List the contents of the current folder for the file system object fsys.  
You can also list the contents of the current folder for the FTP object f.

```
dir(fsys) or dir(f)
4/12/1998    20:00                222390      IO  SYS
 11/2/2003   13:54                  6      MSDOS  SYS
 11/5/1998   20:01                93880  COMMAND  COM
 11/2/2003   13:54    <DIR>           0      TEMP
 11/2/2003   14:00                 33  AUTOEXEC  BAT
  11/2/2003  14:00                 512  BOOTSECT  DOS
  18/2/2003  16:33                4512  SC1SIGNA  DAT
 18/2/2003   16:17    <DIR>           0      FOUND  000
 29/3/2003   19:19                8512      DATA  DAT
 28/3/2003   16:41                8512  DATADATA  DAT
 28/3/2003   16:29                4512  SC4INTEG  DAT
  1/4/2003    9:28           201326592  PAGEFILE  SYS
 11/2/2003   14:13    <DIR>           0      WINNT
  4/5/2001   13:05           214432  NTLDR      '
  4/5/2001   13:05           34468  NTDETECT  COM
 11/2/2003   14:15    <DIR>           0  DRIVERS
  22/1/2001  11:42                217    BOOT    INI '
 28/3/2003   16:41                8512      A      DAT
 29/3/2003   19:19                2512  SC3SIGNA  DAT
 11/2/2003   14:25    <DIR>           0  INETPUB
 11/2/2003   14:28                0    CONFIG  SYS
 29/3/2003   19:10                2512  SC3INTEG  DAT
  1/4/2003   18:05                2512  SC1GAIN  DAT
  11/2/2003  17:26    <DIR>           0  UTILIT~1
```

You must use the `dir(f)` syntax to list the contents of the folder.

## See Also

```
dir | xpctarget.fsbase.mkdir | xpctarget.fsbase.cd |
xpctarget.fsbase.pwd
```

# xpctarget.fsbase.mkdir

---

**Purpose**            Make folder on target computer

**Syntax**            mkdir(file\_obj,dir\_name)  
file\_obj.mkdir(dir\_name)

**Arguments**        file\_obj            Name of the xpctarget.ftp or xpctarget.fs object.  
dir\_name            Name of the folder to be created.

**Description**      Method of xpctarget.fsbase, xpctarget.ftp, and xpctarget.fs objects. From the host computer, makes a new folder in the current folder on the target computer file system.

Note that to delete a folder from the target computer, you need to reboot the computer into DOS or some other operating system and use a utility in that system to delete the folder.

**Examples**          Create a new folder, logs, in the target computer file system object fsys.

```
mkdir(fsys,logs)
```

or

```
fsys.mkdir(logs)
```

Create a new folder, logs, in the target computer FTP object f.  
mkdir(f,logs) or f.mkdir(logs)

**See Also**            mkdir | xpctarget.fsbase.dir | xpctarget.fsbase.pwd



<b>Purpose</b>	Current folder path of target computer
<b>Syntax</b>	<code>pwd(file_obj)</code> <code>file_obj.pwd</code>
<b>Arguments</b>	<code>file_obj</code> Name of the <code>xpctarget.ftp</code> or <code>xpctarget.fs</code> object.
<b>Description</b>	Method of <code>xpctarget.fsbase</code> , <code>xpctarget.ftp</code> , and <code>xpctarget.fs</code> objects. Returns the pathname of the current target computer folder.
<b>Examples</b>	<p>Return the target computer current folder for the file system object <code>fsys</code>.</p> <pre>pwd(fsys) or fsys.pwd</pre> <p>Return the target computer current folder for the FTP object <code>f</code>.</p> <pre>pwd(f) or f.pwd</pre>
<b>See Also</b>	<code>pwd</code>   <code>xpctarget.fsbase.dir</code>   <code>xpctarget.fsbase.mkdir</code>

# xpctarget.fsbase.rmdir

---

**Purpose** Remove folder from target computer

**Syntax** `rmdir(file_obj,dir_name)`  
`file_obj.rmdir(dir_name)`

**Arguments**

<code>dir_name</code>	Name of the folder to remove from the target computer file system.
<code>file_obj</code>	Name of the <code>xpctarget.fs</code> object.

**Description** Method of `xpctarget.fsbase`, `xpctarget.ftp`, and `xpctarget.fs` objects. Removes a folder from the target computer file system.

---

**Note** You cannot recover this folder once it is removed.

---

**Examples** Remove the folder `data2dir.dat` from the target computer file system `fsys`.

```
rmdir(f, 'data2dir.dat')
```

or

```
fsys.rmdir('data2dir.dat')
```

**Purpose** Manage the folders and files on the target computer via file transfer protocol (FTP)

**Description** The FTP object represents the file on the target computer. You work with the file folders using the inherited methods, and transport the file between the host and target computers using the `xpctarget.ftp` methods.

## Constructor

Constructor	Description
<code>xpctarget.ftp</code>	Create file transfer protocol (FTP) object

## Methods

These methods are inherited from `xpctarget.fsbase` Class.

Method	Description
<code>xpctarget.fsbase.cd</code>	Change folder on target computer
<code>xpctarget.fsbase.dir</code>	List contents of current folder on target computer
<code>xpctarget.fsbase.mkdir</code>	Make folder on target computer
<code>xpctarget.fsbase.pwd</code>	Current folder path of target computer
<code>xpctarget.fsbase.rmdir</code>	Remove folder from target computer

These methods are specific to class `ftp`.

Method	Description
<code>xpctarget.ftp.get(ftp)</code>	Retrieve copy of requested file from target computer
<code>xpctarget.ftp.put</code>	Copy file from host computer to target computer

# xpctarget.ftp

---

**Purpose** Create file transfer protocol (FTP) object

**Syntax** `file_object = xpctarget.ftp('mode', 'arg1', 'arg2')`

**Arguments**

<code>file_object</code>	Variable name to reference the FTP object.
<code>mode</code>	Optionally, enter the communication mode:

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

	<code>TCPIP</code>	Specify TCP/IP connection with target computer.
	<code>RS232</code>	Specify RS-232 connection with target computer.
<code>arg1</code>	Optionally, enter an argument based on the <code>mode</code> value:	
	<code>IP address</code>	If <code>mode</code> is 'TCPIP', enter the IP address of the target computer.
	<code>COM port</code>	If <code>mode</code> is 'RS232', enter the host COM port.
<code>arg2</code>	Optionally, enter an argument based on the <code>mode</code> value:	
	<code>Port</code>	If <code>mode</code> is 'TCPIP', enter the port number for the target computer.
	<code>Baud rate</code>	If <code>mode</code> is 'RS232', enter the baud rate for the connection between the host and target computer.

**Description** Constructor of an FTP object (`xpctarget.ftp` Class). The FTP object represents the file on the target computer. You work with the file by changing the file object using methods.

If you have one target computer object, or if you designate a target computer as the default one in your system, use the syntax

```
file_object=xpctarget.ftp
```

If you have multiple target computers in your system, or if you want to identify a target computer with the file object, use the following syntax to create the additional file objects.

```
file_object=xpctarget.ftp('mode', 'arg1', 'arg2')
```

## Examples

In the following example, a file object for a target computer with an RS-232 connection is created.

```
f=xpctarget.ftp('RS232', 'COM1', '115200')
```

```
f =  
xpctarget.ftp
```

Optionally, if you have an `xpctarget.xpc` object, you can construct an `xpctarget.ftp` object by passing the `xpctarget.xpc` object variable to the `xpctarget.ftp` constructor as an argument.

```
>> tg1=xpctarget.xpc('RS232', 'COM1', '115200');  
>> f2=xpctarget.ftp(tg1)
```

```
f2 =  
  
xpctarget.ftp
```

# xpctarget.ftp.get (ftp)

---

**Purpose** Retrieve copy of requested file from target computer

**Syntax** `get(file_obj,file_name)`  
`file_obj.get(file_name)`

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> object.
<code>file_name</code>	Name of a file on the target computer.

**Description** Method of `xpctarget.ftp` objects. Copies the specified filename from the target computer to the current folder of the host computer. `file_name` must be either a fully qualified filename on the target computer, or located in the current folder of the target computer.

**Examples** Retrieve a copy of the file named `data.dat` from the current folder of the target computer file object `f`.

```
get(f,'data.dat') or f.get('data.dat')  
ans = data.dat
```

**See Also** `xpctarget.ftp.put`

**Purpose** Copy file from host computer to target computer

**Syntax** `put(file_obj,file_name)`  
`file_obj.put(file_name)`

**Arguments**

<code>file_obj</code>	Name of the <code>xpctarget.ftp</code> object.
<code>file_name</code>	Name of the file to copy to the target computer.

**Description** Method of `xpctarget.ftp` objects. Copies a file from the host computer to the target computer. `file_name` must be a file in the current folder of the host computer. The method writes `file_name` to the target computer disk.

`put` might be slower than the `get` operation for the same file. This is expected behavior.

**Examples** Copy the file `data2.dat` from the current folder of the host computer to the current folder of the target computer FTP object `f`.

```
put(f, 'data2.dat')
```

or

```
fsys.put('data2.dat')
```

**See Also** `xpctarget.fsbasedir` | `xpctarget.ftp.get (ftp)`

# xpctarget.targets Class

---

**Purpose** Container object to manage target computer environment collection objects

**Description** The targets class contains a collection of environment settings, stored in xpctarget.env Class objects.

## Constructor

Constructor	Description
xpctarget.targets	Create container object to manage target computer environment collection objects

## Methods

Method	Description
xpctarget.targets.Add (env collection object)	Add a new xPC Target environment collection object.
xpctarget.targets.getTargetNames (env collection object)	Retrieve the xPC Target environment collection object names.
xpctarget.targets.Item (env collection object)	Retrieve xPC Target environment collection object.
xpctarget.targets.makeDefault (env collection object)	Set target computer environment collection object as default.
xpctarget.targets.Remove (env collection object)	Remove an xPC Target environment collection object.



## Properties

Property	Description	Writable
DefaultTarget	Returns an <code>xpctarget.env</code> object that references the default target computer object environment.	No
NumTargets	Returns the number of target computer environment objects in the container.	No

# xpctarget.targets

---

**Purpose** Create container object to manage target computer environment collection objects

**Syntax** `env_collection_object = xpctarget.targets`

**Description** Constructor for target environment object collection (`xpctarget.targets` Class). The collection manages the environment object (`xpctarget.env` Class) for a multitarget xPC Target system. (This is in contrast to the `setxpcenv` and `getxpcenv` functions, which manage the environment properties for the default target computer.) You work with the environment objects by changing the environment properties using methods.

Use the syntax

```
env_object = xpctarget.targets
```

Access properties of an `env_collection_object` object with `env_collection_object.propertyname`, `env_collection_object.propertyname.propertyname`, or with the `xpctarget.targets.get (env collection object)` and `xpctarget.targets.set (env collection object)` commands.

Access an individual environment object via `xpctarget.targets.Item (env collection object)`,

**Examples** Create an environment container object. With this object, you can manage the environment collection objects for the targets in your system.

```
tgs=xpctarget.targets
```

**See Also** `xpctarget.targets.get (env collection object)` | `xpctarget.targets.set (env collection object)`

# xpctarget.targets.Add (env collection object)

---

<b>Purpose</b>	Add new xPC Target environment collection object
<b>Syntax</b>	<code>env_collection_object.Add</code>
<b>Description</b>	Method of <code>xpctarget.targets</code> objects. <code>Add</code> creates an xPC Target environment collection object on the host computer.
<b>Examples</b>	<p>Add a new xPC Target environment collection object to the system. Assume that <code>tgs</code> represents the environment collection object. The first <code>get(tgs)</code> function returns the current number of target computers. The second function returns the number of target computers after you add one.</p> <pre>tgs=xpctarget.targets; get(tgs); tgs.Add; get(tgs);</pre>
<b>See Also</b>	<code>xpctarget.targets</code>   <code>xpctarget.targets.set (env collection object)</code>   <code>xpctarget.targets.get (env collection object)</code>

# xpctarget.targets.get (env collection object)

---

**Purpose** Return target object collection environment property values

**Syntax** `get(env_collection_object, 'env_collection_object_property')`

**Arguments**

<code>env_collection_object</code>	Name of a collection of target objects.
<code>'env_collection_object_property'</code>	Name of a target object environment property.

**Description** `get` gets the values of environment properties for a collection of target objects.

The environment properties for a target environment object collection are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Property	Description	Writable
DefaultTarget	Contains an instance of the default target environment object ( <code>xpctarget.env</code> ).	No
NumTargets	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

**Examples** List the values of the target object collection environment property values. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;  
get(tgs);
```

## xpctarget.targets.get (env collection object)

---

List the value for the target object environment collection property NumTargets. Note that the property name is a string, in quotation marks, and not case sensitive.

```
get(tgs, 'NumTargets') or tgs.get('NumTargets')
```

### **See Also**

```
get | xpctarget.targets.set (env collection object) | set
```

# xpctarget.targets.getTargetNames (env collection object)

---

**Purpose** Retrieve xPC Target environment object names

**Syntax** `env_collection_object.getTargetNames`

**Description** Method of `xpctarget.targets` objects. `getTargetNames` retrieves the names of the existing xPC Target environment collection objects from the `xpctarget.targets` class.

**Examples** Retrieve the names of the xPC Target environment collection objects in the system. Assume that `tgs` represents the target object collection environment.

```
tgs=xpctarget.targets;  
get(tgs);  
tgs.getTargetNames
```

**See Also** `xpctarget.targets` | `xpctarget.targets.set (env collection object)` | `xpctarget.targets.get (env collection object)`

# xpctarget.targets.Item (env collection object)

---

<b>Purpose</b>	Retrieve specific xPC Target environment (env) object
<b>Syntax</b>	<code>env_collection_object.Item('env_object_name')</code>
<b>Description</b>	Method of <code>xpctarget.targets</code> objects. <code>Item</code> retrieves a specific environment object ( <code>xpctarget.env Class</code> ) from the <code>xpctarget.targets</code> class. Use this method to work with a particular target computer environment object.
<b>Examples</b>	<p>Retrieve a new xPC Target environment collection object from the system. Assume that <code>tgs</code> represents the target object collection environment.</p> <pre>tgs=xpctarget.targets; get(tgs); tgs.getTargetNames tgs.Item('TargetPC1')</pre>
<b>See Also</b>	<code>xpctarget.targets</code>   <code>xpctarget.targets.set (env collection object)</code>   <code>xpctarget.targets.get (env collection object)</code>

# xpctarget.targets.makeDefault (env collection object)

---

<b>Purpose</b>	Set specific target computer environment object as default
<b>Syntax</b>	<code>env_collection_object.makeDefault('env_object_name')</code>
<b>Description</b>	Method of <code>xpctarget.targets</code> objects. <code>makeDefault</code> sets the specified target computer environment object as the default target computer from the <code>xpctarget.targets</code> class.
<b>Examples</b>	<p>Set the specified target collection object as the default target computer collection. Assume that <code>tgs</code> represents the target object collection environment.</p> <pre>tgs=xpctarget.targets; get(tgs); tgs.getTargetNames tgs.makeDefault('TargetPC2')</pre>
<b>See Also</b>	<code>xpctarget.targets</code>   <code>xpctarget.targets.set (env collection object)</code>   <code>xpctarget.targets.get (env collection object)</code>



# xpctarget.targets.Remove (env collection object)

---

<b>Purpose</b>	Remove specific xPC Target environment object
<b>Syntax</b>	<code>env_collection_object.Remove('env_collection_object_name')</code>
<b>Description</b>	Method of <code>xpctarget.targets</code> objects. Remove removes an existing xPC Target environment object from the environment collection. If you remove the target environment object of the default target computer, the next target environment object becomes the default target computer. You can remove all but the last target computer, which becomes the default target computer.
<b>Examples</b>	<p>Remove an xPC Target environment collection object from the system. Assume that <code>tgs</code> represents the target object collection environment.</p> <pre>tgs=xpctarget.targets; get(tgs); tgs.getTargetNames tgs.Remove('TargetPC2')</pre>
<b>See Also</b>	<code>xpctarget.targets</code>   <code>xpctarget.targets.set (env collection object)</code>   <code>xpctarget.targets.get (env collection object)</code>

# xpctarget.targets.set (env collection object)

---

**Purpose** Change target object environment collection object property values

**Syntax**

```
set(env_collection_object)
set(env_collection_object, 'property_name1',
'property_value1','property_name2', 'property_value2', . . .)
env_collection_object.set('property_name1',
'property_value1')
set(env_collection_object, property_name_vector,
property_value_vector)
env_collection_object.property_name = property_value
```

**Arguments**

<code>env_collection_object</code>	Name of a target environment collection object.
<code>'property_name'</code>	Name of a target object environment collection property. Always use quotation marks for character strings.
<code>property_value</code>	Value for a target object environment collection property. Always use quotation marks for character strings; quotation marks are optional for numbers.

**Description** `set` sets the values of environment properties for a collection of target object environments. Not all properties are user writable. Properties are entered as property-value pairs.

The environment properties for a target object collection are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

## xpctarget.targets.set (env collection object)

---

Property	Description	Writable
DefaultTarget	Contains an instance of the default target environment object (xpctarget.env).	No
NumTargets	Contains the number of target objects in the xPC Target system. Note that this is not the actual number of target computers in the system.	No

### See Also

get | set | xpctarget.targets.get (env collection object)

# xpctarget.xpc Class

---

**Purpose** Target object representing target application

**Description** Provides access to methods and properties used to start and stop the target application, read and set parameters, monitor signals, and retrieve status information about the target computer.

## Constructor

Constructor	Description
xpctarget.xpc	Create target object representing target application

## Methods

Method	Description
xpctarget.xpc.addscope	Create scopes
xpctarget.xpc.close	Close serial port connecting host computer with target computer
xpctarget.xpc.get (target application object)	Return target application object property values
xpctarget.xpc.getlog	All or part of output logs from target object
xpctarget.xpc.getparam	Value of target object parameter index
xpctarget.xpc.getparamlist	Parameter index from parameter list
xpctarget.xpc.getparamname	Block path and parameter name from index list
xpctarget.xpc.getscope	Scope object pointing to scope defined in kernel
xpctarget.xpc.getsignal	Value of target object signal index
xpctarget.xpc.getsignalname	Signal index or signal property from signal list
xpctarget.xpc.getsignalnames	Return labels of signal indices
xpctarget.xpc.getsignalvalue	Return signal label
xpctarget.xpc.getsignalvaluefromlabel	Signal name from index list

Method	Description
<code>xpctarget.xpc.getxpcpci</code>	Determine which PCI boards are installed in target computer
<code>xpctarget.xpc.load</code>	Download target application to target computer
<code>xpctarget.xpc.loadparameters</code>	Restore parameter values saved in specified file
<code>xpctarget.xpc.reboot</code>	Reboot target computer
<code>xpctarget.xpc.remscope</code>	Remove scope from target computer
<code>xpctarget.xpc.saveparameters</code>	Save current target application parameter values
<code>xpctarget.xpc.set (target application object)</code>	Change target application object property values
<code>xpctarget.xpc.setparameter</code>	Change writable target object parameters
<code>xpctarget.xpc.start (target application object)</code>	Start execution of target application on target computer
<code>xpctarget.xpc.stop (target application object)</code>	Stop execution of target application on target computer
<code>xpctarget.xpc.targetping</code>	Test communication between host and target computers
<code>xpctarget.xpc.unload</code>	Remove current target application from target computer

## Properties

Properties are read using `xpctarget.xpc.get (target application object)`. Writable properties are written using `xpctarget.xpc.set (target application object)`.

# xpctarget.xpc Class

Property	Description	Writable
Application	Name of the Simulink model and target application built from that model.	No
AvgTET	<p>Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution.</p> <p>The TET includes:</p> <ul style="list-style-type: none"> <li>• Complete I/O latency.</li> <li>• Data logging (the parts that happen in a real-time task). This includes data captured in scopes.</li> <li>• Asynchronous interruptions.</li> <li>• Parameter updating latency (if the <b>Double buffer parameter changes</b> parameter is set in the <b>xPC Target options</b> node of the model Configuration Parameters dialog box).</li> </ul> <p>Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are:</p> <ul style="list-style-type: none"> <li>• Time required to measure TET</li> <li>• Interrupt latency required to schedule and run one step of the model</li> </ul>	No

Property	Description	Writable
CommunicationTimeout	Communication timeout between host and target computer, in seconds.	Yes
Connected	Communication status between the host computer and the target computer. Values are 'Yes' and 'No'.	No
CPUoverload	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from 'none' to 'detected' and the current run is stopped. Returning this status to 'none' requires either a faster processor or a larger sample time.	No
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No
LogMode	Controls which data points are logged: <ul style="list-style-type: none"><li>• Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li><li>• Value-equidistant logging. Logs a data point only when an output signal from the <code>OutputLog</code> changes by a specified value (increment). Set the value to the difference in signal values.</li></ul>	Yes

## xpctarget.xpc Class

Property	Description	Writable
MaxLogSamples	<p>Maximum number of samples for each logged signal within the circular buffers for TimeLog, StateLog, OutputLog, and TETLog. StateLog and OutputLog can have one or more signals.</p> <p>This value is calculated by dividing the <b>Signal Logging Buffer Size</b> by the number of logged signals. The <b>Signal Logging Buffer Size</b> box is in the <b>xPC Target options</b> pane of the Configuration Parameters dialog box.</p>	No
MaxTET	<p>Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.</p>	No
MinTET	<p>Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.</p>	No
Mode	<p>Type of Simulink Coder™ code generation. Values are 'Real-Time Singletasking', 'Real-Time Multitasking', and 'Accelerate'. The default value is 'Real-Time Singletasking'. Even if you select 'Real-Time Multitasking', the actual mode can be 'Real-Time Singletasking'. This happens if your model contains only one or two tasks and the sample rates are equal.</p>	No



Property	Description	Writable
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds MaxLogSamples.	No
NumParameters	The number of parameters from your Simulink model that you can tune or change.	No
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	No
OutputLog	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No
Parameters	List of tunable parameters. This list is visible only when ShowParameters is set to 'on': <ul style="list-style-type: none"><li>• Property value. Value of the parameter in a Simulink block.</li><li>• Type. Data type of the parameter. Always double.</li><li>• Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix.</li><li>• Parameter name. Name of a parameter in a Simulink block.</li><li>• Block name. Name of a Simulink block.</li></ul>	No

## xpctarget.xpc Class

---

Property	Description	Writable
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “User Interaction” for limitations on target property changes to sample times.)	Yes
Scopes	List of index numbers, with one index for each scope.	No
SessionTime	Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.	No
ShowParameters	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
ShowSignals	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
Signals	List of viewable signals. This list is visible only when ShowSignals is set to 'on'. <ul style="list-style-type: none"><li>• Property name. S0, S1. . .</li><li>• Property value. Value of the signal.</li><li>• Block name. Name of the Simulink block the signal is from.</li></ul>	No
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No

Property	Description	Writable
Status	Execution status of your target application. Values are 'stopped' and 'running'.	No
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the <b>Solver</b> pane of the Configuration Parameters dialog box.  When the ExecTime reaches StopTime, the application stops running.	Yes
TETLog	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application.  To enable logging of the TET, you need to select the <b>Log Task Execution Time</b> check box in the <b>xPC Target options</b> pane of the Configuration Parameters dialog box.	No
TimeLog	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

# xpctarget.xpc

---

**Purpose** Create target object representing target application

**Syntax**  
target\_object = xpctarget.xpc('mode', 'arg1', 'arg2')  
target\_object=xpctarget.xpc('target\_object\_name')

**Arguments**

target_object	Variable name to reference the target object
mode	Optionally, enter the communication mode

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**TCP/IP** Enable TCP/IP connection with target computer.

**RS232** Enable RS-232 connection with target computer.

arg1 Optionally, enter an argument based on the mode value:

**IP** If mode is 'TCP/IP', enter the IP address of the target computer.

**COM** If mode is 'RS232', enter the host port COM port.

arg2 Optionally, enter an argument based on the mode value:

**Port** If mode is 'TCP/IP', enter the port number for the target computer.

Baud rate      If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

target\_object\_name Target object name as specified in the xPC Target Explorer

## Description

Constructor of a target object (xpctarget.xpc Class). The target object represents the target application and target computer. You make changes to the target application by changing the target object using methods and properties.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
target_object=xpctarget.xpc
```

If you have multiple target computers in your system, use the following syntax to create the additional target objects.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax to construct a corresponding target object from the MATLAB Command Window.

```
target_object=xpctarget.xpc('target_object_name')
```

## Examples

Before you build a target application, you can check the connection between your host and target computers by creating a target object, then using the xpctarget.xpc.targetping method to check the connection.

```
tg = xpctarget.xpc
xPC Object
    Connected                    = Yes
    Application                 = loader
```

```
tg.targetping
```

```
ans =  
  
success
```

If you have a second target computer for which you want to check the connection, create a second target object. In the following example, the connection with the second target computer is an RS-232 connection.

```
tg1=xpctarget.xpc('RS232','COM1','115200')  
  
xPC Object  
  Connected          = Yes  
  Application        = loader
```

If you have an xPC Target Explorer target object, and you want to construct a corresponding target object in the MATLAB Command Window, use a command like the following:

```
target_object=xpctarget.xpc('TargetPC1')
```

### See Also

```
xpctarget.xpc.get (target application object) |  
xpctarget.xpc.set (target application object) |  
xpctarget.xpc.targetping
```

## Purpose

Create scopes

## Syntax

Create a scope and scope object without assigning to a MATLAB variable.

```
addscope(target_object, scope_type, scope_number)  
target_object.addscope(scope_type, scope_number)
```

Create a scope, scope object, and assign to a MATLAB variable

```
scope_object = addscope(target_object,  
    scope_type, scope_number)  
scope_object = target_object.addscope(scope_type,  
    scope_number)
```

**Target computer command line** — When you are using this command on the target computer, you can only add a target scope.

```
addscope  
addscope scope_number
```

## Arguments

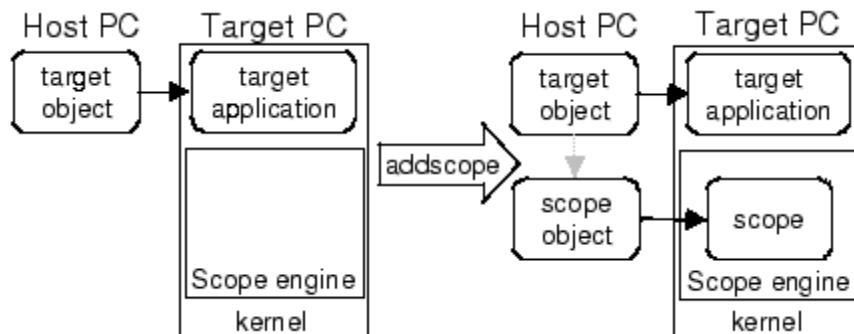
- target\_object** Name of a target object. The default target name is `tg`.
- scope\_type** Values are `'host'`, `'target'`, or `'file'`. This argument is optional with `host` as the default value.
- scope\_number** Vector of new scope indices. This argument is optional. The next available integer in the target object property `Scopes` as the default value.
- If you enter a scope index for an existing scope object, the result is an error.

## Description

`addscope` creates a scope of the specified type and updates the target object property `Scopes`. This method returns a scope object vector. If the result is not assigned to a variable, the scope object properties are listed in the MATLAB window. The xPC Target product supports 10

# xpctarget.xpc.addscope

target or host scopes, and eight file scopes, for a maximum of 28 scopes. If you try to add a scope with the same index as an existing scope, the result is an error.



## Examples

Create a scope and scope object `sc1` using the method `addscope`. A target scope is created on the target computer with an index of 1, and a scope object is created on the host computer, assigned to the variable `sc1`. The target object property `Scopes` is changed from `No scopes defined` to 1.

```
sc1 = addscope(tg, 'target', 1)
```

or

```
sc1 = tg.addscope('target', 1)
```

Create a scope with the method `addscope` and then create a scope object, corresponding to this scope, using the method `getscope`. A target scope is created on the target computer with an index of 1, and a scope object is created on the host computer, but it is not assigned to a variable. The target object property `Scopes` is changed from `No scopes defined` to 1.

```
addscope(tg, 'target', 1) or tg.addscope('target', 1)  
sc1 = getscope(tg, 1) or sc1 = tg.getscope(1)
```



Create two scopes using a vector of scope objects `scvector`. Two target scopes are created on the target computer with scope indices of 1 and 2, and two scope objects are created on the host computer that represent the scopes on the target computer. The target object property `Scopes` is changed from `No scopes defined` to `1,2`.

```
scvector = addscope(tg, 'target', [1, 2])
```

Create a scope and scope object `sc4` of type `file` using the method `addscope`. A file scope is created on the target computer with an index of 4. A scope object is created on the host computer and is assigned to the variable `sc4`. The target object property `Scopes` is changed from `No scopes defined` to `4`.

```
sc4 = addscope(tg,'file',4) or sc4 = tg.addscope('file',4)
```

## See Also

[xpctarget.xpc.remscope](#) | [xpctarget.xpc.getscope](#)

## How To

- “Application and Driver Scripts”

## xpctarget.xpc.close

---

**Purpose** Close serial port connecting host computer with target computer

**Syntax** `close(target_object)`  
`target_object.close`

**Arguments** `target_object` Name of a target object.

**Description** `close` closes the serial connection between the host computer and a target computer. If you want to use the serial port for another function without quitting the MATLAB window – for example, a modem – use this function to close the connection.

# xpctarget.xpc.get (target application object)

**Purpose** Return target application object property values

**Syntax** `get(target_object, 'target_object_property')`

**Arguments**

<code>target_object</code>	Name of a target object.
<code>'target_object_property'</code>	Name of a target object property.

**Description** `get` gets the value of readable target object properties from a target object.

The properties for a target object are listed in the following table. This table includes a description of the properties and which properties you can change directly by assigning a value.

Property	Description	Writable
Application	Name of the Simulink model and target application built from that model.	No
AvgTET	<p>Average task execution time. This value is an average of the measured CPU times, in seconds, to run the model equations and post outputs during each sample interval. Task execution time is nearly constant, with minor deviations due to cache, memory access, interrupt latency, and multirate model execution.</p> <p>The TET includes:</p> <ul style="list-style-type: none"><li>• Complete I/O latency.</li><li>• Data logging (the parts that happen in a real-time task). This includes data captured in scopes.</li><li>• Asynchronous interruptions.</li></ul>	No

## xpctarget.xpc.get (target application object)

Property	Description	Writable
	<ul style="list-style-type: none"> <li>Parameter updating latency (if the <b>Double buffer parameter changes</b> parameter is set in the <b>xPC Target options</b> node of the Configuration Parameters dialog box).</li> </ul> <p>Note that the TET is not the only consideration in determining the minimum achievable sample time. Other considerations, not included in the TET, are:</p> <ul style="list-style-type: none"> <li>Time required to measure TET</li> <li>Interrupt latency required to schedule and run one step of the model</li> </ul>	
CommunicationTimeout	Communication timeout between host and target computer, in seconds.	Yes
Connected	Communication status between the host computer and the target computer. Values are 'Yes' and 'No'.	No
CPUoverload	CPU status for overload. If the target application requires more CPU time than the sample time of the model, this value is set from 'none' to 'detected' and the current run is stopped. Returning this status to 'none' requires either a faster processor or a larger sample time.	No
ExecTime	Execution time. Time, in seconds, since your target application started running. When the target application stops, the total execution time is displayed.	No

## xpctarget.xpc.get (target application object)

Property	Description	Writable
LogMode	Controls which data points are logged: <ul style="list-style-type: none"><li>• Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li><li>• Value-equidistant logging. Logs a data point only when an output signal from the OutputLog changes by a specified value (increment). Set the value to the difference in signal values.</li></ul>	Yes
MaxLogSamples	Maximum number of samples for each logged signal within the circular buffers for TimeLog, StateLog, OutputLog, and TETLog. StateLog and OutputLog can have one or more signals.  This value is calculated by dividing the <b>Signal Logging Buffer Size</b> by the number of logged signals. The <b>Signal Logging Buffer Size</b> box is in the <b>xPC Target options</b> pane of the Configuration Parameters dialog box.	No
MaxTET	Maximum task execution time. Corresponds to the slowest time (longest time measured), in seconds, to update model equations and post outputs.	No
MinTET	Minimum task execution time. Corresponds to the fastest time (smallest time measured), in seconds, to update model equations and post outputs.	No

## xpctarget.xpc.get (target application object)

---

Property	Description	Writable
Mode	Type of Simulink Coder code generation. Values are 'Real-Time Singletasking', 'Real-Time Multitasking', and 'Accelerate'. The default value is 'Real-Time Singletasking'. Even if you select 'Real-Time Multitasking', the actual mode can be 'Real-Time Singletasking'. This happens if your model contains only one or two tasks and the sample rates are equal.	No
NumLogWraps	The number of times the circular buffer wrapped. The buffer wraps each time the number of samples exceeds MaxLogSamples.	No
NumParameters	The number of parameters from your Simulink model that you can tune or change.	No
NumSignals	The number of signals from your Simulink model that are available to be viewed with a scope.	No
OutputLog	Storage in the MATLAB workspace for the output or Y-vector logged during execution of the target application.	No

## xpctarget.xpc.get (target application object)

Property	Description	Writable
Parameters	<p>List of tunable parameters. This list is visible only when ShowParameters is set to 'on':</p> <ul style="list-style-type: none"><li>• Property value. Value of the parameter in a Simulink block.</li><li>• Type. Data type of the parameter. Always double.</li><li>• Size. Size of the parameter. For example, scalar, 1-by-2 vector, or 2-by-3 matrix.</li><li>• Parameter name. Name of a parameter in a Simulink block.</li><li>• Block name. Name of a Simulink block.</li></ul>	No
SampleTime	<p>Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. (See “User Interaction” for limitations on target property changes to sample times.)</p>	Yes
Scopes	<p>List of index numbers, with one index for each scope.</p>	No
SessionTime	<p>Time since the kernel started running on your target computer. This is also the elapsed time since you booted the target computer. Values are in seconds.</p>	No
ShowParameters	<p>Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.</p>	Yes

## xpctarget.xpc.get (target application object)

---

Property	Description	Writable
ShowSignals	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
Signals	List of viewable signals. This list is visible only when ShowSignals is set to 'on'. <ul style="list-style-type: none"><li>• Property name. S0, S1. . .</li><li>• Property value. Value of the signal.</li><li>• Block name. Name of the Simulink block the signal is from.</li></ul>	No
StateLog	Storage in the MATLAB workspace for the state or x-vector logged during execution of the target application.	No
Status	Execution status of your target application. Values are 'stopped' and 'running'.	No
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the <b>Solver</b> pane of the Configuration Parameters dialog box. When the ExecTime reaches StopTime, the application stops running.	Yes



## xpctarget.xpc.get (target application object)

Property	Description	Writable
TETLog	Storage in the MATLAB workspace for a vector containing task execution times during execution of the target application.  To enable logging of the TET, you need to select the <b>Log Task Execution Time</b> check box in the <b>xPC Target options</b> pane of the Configuration Parameters dialog box.	No
TimeLog	Storage in the MATLAB workspace for the time or T-vector logged during execution of the target application.	No
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

### Examples

List the value for the target object property StopTime. Notice that the property name is a string, in quotation marks, and not case sensitive.

```
get(tg,'stoptime') or tg.get('stoptime')  
ans = 0.2
```

### See Also

```
get | set | xpctarget.xpc.set (target application object)  
| xpctarget.xpcsc.get (scope object) | xpctarget.xpc.set  
(target application object)
```

# xpctarget.xpc.getlog

---

**Purpose** All or part of output logs from target object

**Syntax** `log = getlog(target_object, 'log_name', first_point, number_samples, decimation)`

**Arguments**

<code>log</code>	User-defined MATLAB variable.
<code>'log_name'</code>	Values are TimeLog, StateLog, OutputLog, or TETLog. This argument is required.
<code>first_point</code>	First data point. The logs begin with 1. This argument is optional. Default is 1.
<code>number_samples</code>	Number of samples after the start time. This argument is optional. Default is all points in log.
<code>decimation</code>	1 returns all sample points. n returns every nth sample point. This argument is optional. Default is 1.

**Description** Use this function instead of the function `get` when you want only part of the data.

**Examples** To get the first 1000 points in a log,

```
Out_log = getlog(tg, 'TETLog', 1, 1000)
```

To get every other point in the output log and plot values,

```
Output_log = getlog(tg, 'TETLog', 1, 10, 2)
Time_log = getlog(tg, 'TimeLog', 1, 10, 2)
plot(Time_log, Output_log)
```

**How To**

- `xpctarget.xpc.get` (target application object)
- “Set Configuration Parameters”

**Purpose** Value of target object parameter index

**Syntax** `getparam(target_object, parameter_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>parameter_index</code>	Index number of the parameter.

**Description** `getparam` returns the value of the parameter associated with `parameter_index`.

**Examples** Get the value of parameter index 5.

```
getparam(tg, 5)
ans = 400
```

# xpctarget.xpc.getparamid

---

**Purpose** Parameter index from parameter list

**Syntax** `getparamid(target_object, 'block_name', 'parameter_name')`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>'block_name'</code>	Simulink block path without model name.
<code>'parameter_name'</code>	Name of a parameter within a Simulink block.

**Description** `getparamid` returns the index of a parameter in the parameter list based on the path to the parameter name. The names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Simulink Coder uses for code generation.

**Examples** Get the parameter property for the parameter `Gain` in the Simulink block `Gain1`, incrementally increase the gain, and pause to observe the signal trace.

```
id = getparamid(tg, 'Subsystem/Gain1', 'Gain')
for i = 1 : 3
    set(tg, id, i*2000);
    pause(1);
end
```

Get the property index of a single block.

```
getparamid(tg, 'Gain1', 'Gain') ans = 5
```

**See Also** `xpctarget.xpc.getsignalid`

**How To**

- “Application and Driver Scripts”
- “Why Does the `getparamid` Function Return Nothing?”

**Purpose** Block path and parameter name from index list

**Syntax** `getparamname(target_object, parameter_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>parameter_index</code>	Index number of the parameter.

**Description** `getparamname` returns two argument strings, block path and parameter name, from the index list for the specified parameter index.

**Examples** Get the block path and parameter name of parameter index 5.

```
[blockPath,parName]=getparamname(tg,5)
blockPath =
Signal Generator
parName =
Amplitude
```

# xpc target.xpc.getscope

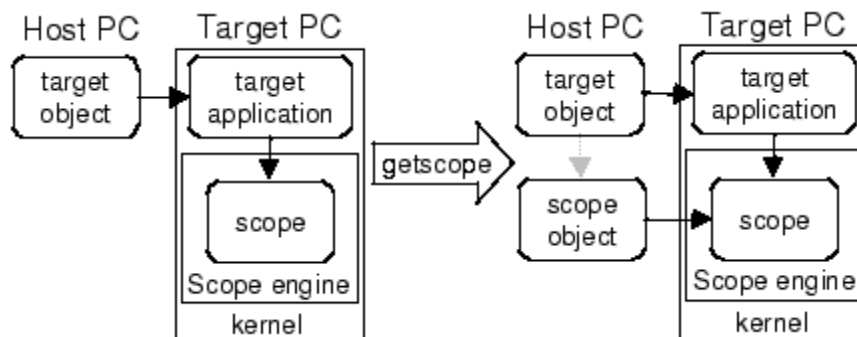
**Purpose** Scope object pointing to scope defined in kernel

**Syntax**  
`scope_object_vector = getscope(target_object, scope_number)`  
`scope_object = target_object.getscope(scope_number)`

**Arguments**

<code>target_object</code>	Name of a target object.
<code>scope_number_vector</code>	Vector of existing scope indices listed in the target object property <code>Scopes</code> . The vector can have only one element.
<code>scope_object</code>	MATLAB variable for a new scope object vector. The vector can have only one scope object.

**Description** `getscope` returns a scope object vector. If you try to get a nonexistent scope, the result is an error. You can retrieve the list of existing scopes using the method `get(target_object, 'scopes')` or `target_object.scopes`.



**Examples** If your Simulink model has an xPC Target scope block, a target scope is created at the time the target application is downloaded to the target computer. To change the number of samples, you need to create a scope object and then change the scope object property `NumSamples`.

```
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
sc1.NumSample = 500
```

The following example gets the properties of all scopes on the target computer and creates a vector of scope objects on the host computer. If the target object has more than one scope, it create a vector of scope objects.

```
scvector = getscope(tg)
```

### **See Also**

[getxpcenv](#) | [xpctarget.xpc.remscope](#)

### **How To**

- “Application and Driver Scripts”

# xpctarget.xpc.getsignal

---

**Purpose** Value of target object signal index

**Syntax** `getsignal(target_object, signal_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>signal_index</code>	Index number of the signal.

**Description** `getsignal` returns the value of the signal associated with `signal_index`.

**Examples** Get the value of signal index 2.

```
getsignal(tg, 2)
ans = -3.3869e+006
```



**Purpose**

Signal index or signal property from signal list

**Syntax**

```
getsignalid(target_object, 'signal_name')  
tg.getsignalid('signal_name')
```

**Arguments**

<code>target_object</code>	Name of an existing target object.
<code>signal_name</code>	Enter the name of a signal from your Simulink model. For blocks with a single signal, the <code>signal_name</code> is equal to the <code>block_name</code> . For blocks with multiple signals, the xPC Target software appends S1, S2 . . . to the <code>block_name</code> .

**Description**

`getsignalid` returns the index or name of a signal from the signal list, based on the path to the signal name. The block names must be entered in full and are case sensitive. Note, enter for `block_name` the mangled name that Simulink Coder uses for code generation.

**Examples**

Get the signal index for the single signal from the Simulink block Gain1.

```
getsignalid(tg, 'Gain1') or tg.getsignalid('Gain1')  
ans = 6
```

**See Also**

`xpctarget.xpc.getparamid`

**How To**

- “Application and Driver Scripts”
- “Why Does the `getparamid` Function Return Nothing?”

# xpctarget.xpc.getsignalidsfromlabel

---

**Purpose** Return vector of signal indices

**Syntax** `getsignalidsfromlabel(target_object, signal_label)`  
`target_object.getsignalidsfromlabel(signal_label)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>signal_label</code>	Signal label (from Simulink model).

**Description** `getsignalidsfromlabel` returns a vector of one or more signal indices that are associated with the labeled signal, `signal_label`. This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels.

**Examples** Get the vector of signal indices for a signal labeled Gain.

```
>> tg.getsignalidsfromlabel('xpcoscGain')  
ans =  
0
```

**See Also** `xpctarget.xpc.getsignallabel`

**Purpose** Return signal label

**Syntax** `getsignallabel(target_object, signal_index)`  
`target_object.getsignallabel(signal_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is tg.
<code>signal_index</code>	Index number of the signal.

**Description** `getsignallabel` returns the signal label for the specified signal index, `signal_index`. `signal_label`. This function assumes that you have labeled the signal for which you request the label (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels.

**Examples**

```
>> getsignallabel(tg, 0)
ans =
xpcoscGain
```

**See Also** `xpctarget.xpc.getsignalidsfromlabel`

# xpctarget.xpc.getsignalname

---

**Purpose** Signal name from index list

**Syntax** `getsignalname(target_object, signal_index)`  
`target_object.getsignalname(signal_index)`

**Arguments**

<code>target_object</code>	Name of a target object. The default name is <code>tg</code> .
<code>signal_index</code>	Index number of the signal.

**Description** `getparamname` returns one argument string, signal name, from the index list for the specified signal index.

**Examples** Get the signal name of signal ID 2.

```
[sigName]=getsignalname(tg,2)
sigName =
Gain2
```

**Purpose** Determine which PCI boards are installed in target computer

**Syntax** `getxpcpci(target_object, 'type_of_boards')`  
`getxpcpci(target_object, 'verbose')`

**Arguments**

<code>target_object</code>	Variable name to reference the target object.
<code>type_of_boards</code>	Values are no arguments, 'all', and 'supported'.
<code>verbose</code>	Argument to include the base address register information in the PCI device display.

**Description** The `getxpcpci` function displays, in the MATLAB window, which PCI boards are installed in the target computer. By default, `getxpcpci` displays this information for the target object, `tg`. If you have multiple target computers in your system, you can call the `getxpcpci` function for a particular target object, `target_object`.

Only devices supported by driver blocks in the xPC Target block library are displayed. The information includes the PCI bus number, slot number, assigned IRQ number, manufacturer name, board name, device type, manufacturer PCI ID, base address, and the board PCI ID itself.

The following preconditions must be met before you can use this function:

- The host-target communication link must be working. (The function `xpctargetping` must return `success` before you can use the function `getxpcpci`.)
- Either a target application is loaded or the loader is active. The latter is used to query for resources assigned to a specific PCI device, which have to be provided to a driver block dialog box before the model build process.

## Examples

The following example displays the installed PCI devices, not only the devices supported by the xPC Target block library. This includes graphics controllers, network cards, SCSI cards, and even devices that are part of the motherboard chip set (for example, PCI-to-PCI bridges).

```
getxpcpci('all')
```

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information.

```
getxpcpci('supported')
```

The following example displays a list of the currently supported PCI devices in the xPC Target block library, including subvendor and subdevice information and base address register contents.

```
getxpcpci('verbose')
```

When called with the 'supported' option, `getxpcpci` does not access the target computer.

To display the list of PCI devices installed on the target computer, `tg1`, first create a target object, `tg1`, for that target computer. Then, call `getxpcpci` with the 'all' option. For example:

```
tg1=xpctarget.xpc('RS232','COM1','115200')
getxpcpci(tg1, 'all')
```

To return the result of a `getxpcpci` query in the struct `pcidevs` instead of displaying it, assign the function to `pcidevs`. The struct `pcidevs` is an array with one element for each detected PCI device. Each element combines the information by a set of field names. The struct contains more information compared to the displayed list. Its contents vary according to the options you specify for the function.

```
pcidevs = getxpcpci
```

**Purpose** Download target application to target computer

**Syntax** `load(target_object, 'target_application')`  
`target_object.load('target_application')`

**Arguments**

<code>target_object</code>	Name of an existing target object.
<code>target_application</code>	Simulink model and target application name.

**Description** Before using this function, the target computer must be booted with the xPC Target kernel, and the target application must be built in the current working folder on the host computer.

If an application was previously loaded, the old target application is first unloaded before downloading the new target application. The method `load` is called automatically after the Simulink Coder build process.

---

**Note** If you are running in Standalone mode, this command has no effect. To load a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

---

**Examples** Load the target application `xpcosc` represented by the target object `tg`.

```
load(tg, 'xpcosc') or tg.load('xpcosc')
+tg or tg.start or start(tg)
```

**See Also** `xpctarget.xpc.unload`

**How To**

- “Application and Driver Scripts”

# xpctarget.xpc.loadparamset

---

**Purpose** Restore parameter values saved in specified file

**Syntax** `loadparamset(target_object, 'filename')`  
`target_object.loadparamset('filename')`

**Arguments**

<code>target_object</code>	Name of an existing target object.
<code>filename</code>	Enter the name of the file that contains the saved parameters.

**Description** `loadparamset` restores the target application parameter values saved in the file `filename`. This file must be located on a local drive of the target computer. This method assumes that you have a parameter file from a previous run of the `xpctarget.xpc.saveparamset` method.

**See Also** `xpctarget.xpc.saveparamset`



**Purpose** Reboot target computer

**Syntax** MATLAB command line  
`reboot(target_object)`

**Target computer command line**

`reboot`

**Arguments** `target_object` Name of an existing target object.

**Description** `reboot` reboots the target computer, and if a target boot disk is still present, the xPC Target kernel is reloaded.  
On the target computer command line, you can use the corresponding command `reboot`.  
You can also use this method to reboot the target computer back to Windows after removing the target boot disk.

---

**Note** This method might not work on some target hardware.

---

**See Also** `xpctarget.xpc.load` | `xpctarget.xpc.unload`

# xpctarget.xpc.remscope

**Purpose** Remove scope from target computer

**Syntax** MATLAB command line

```
remscope(target_object, scope_number_vector)
target_object.remscope(scope_number_vector)
remscope(target_object)
target_object.remscope
```

**Target computer command line**

```
remscope scope_number
remscope 'all'
```

## Arguments

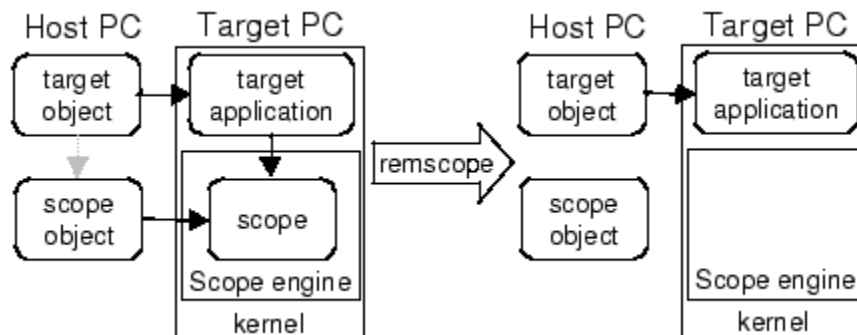
**target\_object** Name of a target object. The default name is tg.

**scope\_number\_vector** Vector of existing scope indices listed in the target object property Scopes.

**scope\_number** Single scope index.

## Description

If a scope index is not given, the method `remscope` deletes all scopes on the target computer. The method `remscope` has no return value. The scope object representing the scope on the host computer is not deleted.



Note that you can only permanently remove scopes that are added with the method `addscope`. This is a scope that is outside a model. If you remove a scope that has been added through a scope block (the scope block is inside the model), a subsequent run of that model creates the scope again.

## Examples

Remove a single scope.

```
remscope(tg, 1)
```

or

```
tg.remscope(1)
```

Remove two scopes.

```
remscope(tg, [1 2])
```

or

```
tg.remscope([1,2])
```

Remove all scopes.

```
remscope(tg)
```

or

```
tg.remscope
```

## See Also

`xpctarget.xpc.addscope` | `xpctarget.xpc.getscope`

## How To

- “Application and Driver Scripts”

# xpctarget.xpc.saveparamset

---

**Purpose** Save current target application parameter values

**Syntax** `saveparamset(target_object, 'filename')`  
`target_object.saveparamset('filename')`

**Arguments**

<code>target_object</code>	Name of an existing target object.
<code>filename</code>	Enter the name of the file to contain the saved parameters.

**Description** `saveparamset` saves the target application parameter values in the file `filename`. This method saves the file on a local drive of the target computer (C:\ by default). You can later reload these parameters with the `xpctarget.xpc.loadparamset` function.

You might want to save target application parameter values if you change these parameter values while the application is running in real time. Saving these values enables you to easily recreate target application parameter values from a number of application runs.

**See Also** `xpctarget.xpc.loadparamset`

# xpctarget.xpc.set (target application object)

---

**Purpose** Change target application object property values

**Syntax** MATLAB command line

```
set(target_object)
set(target_object, 'property_name1', 'property_value1',
'property_name2', 'property_value2', . . .)
target_object.set('property_name1', 'property_value1')
set(target_object, property_name_vector,
property_value_vector)
target_object.property_name = property_value
```

**Target computer command line** - Commands are limited to the target object properties stoptime, sampletime, and parameters.

```
parameter_name = parameter_value
stoptime = floating_point_number
sampletime = floating_point_number
```

## Arguments

target_object	Name of a target object.
'property_name'	Name of a target object property. Always use quotation marks.
property_value	Value for a target object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

## Description

set sets the properties of the target object. Not all properties are user writable.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in property\_name\_vector are stored in property\_value\_vector. The writable properties for a target object

## xpctarget.xpc.set (target application object)

---

are listed in the following table. This table includes a description of the properties:

Property	Description	Writable
CommunicationTimeout	Communication timeout between host and target computer, in seconds.	Yes
LogMode	Controls which data points are logged: <ul style="list-style-type: none"><li>• Time-equidistant logging. Logs a data point at every time interval. Set value to 'Normal'.</li><li>• Value-equidistant logging. Logs a data point only when an output signal from the OutputLog changes by a specified value (increment). Set the value to the difference in signal values.</li></ul>	Yes
SampleTime	Time between samples. This value equals the step size, in seconds, for updating the model equations and posting the outputs. See “User Interaction” for limitations on target property changes to sample times.	Yes

## xpctarget.xpc.set (target application object)

---

Property	Description	Writable
ShowParameters	Flag set to view or hide the list of parameters from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
ShowSignals	Flag set to view or hide the list of signals from your Simulink blocks. This list is shown when you display the properties for a target object. Values are 'on' and 'off'.	Yes
StopTime	Time when the target application stops running. Values are in seconds. The original value is set in the <b>Solver</b> pane of the Configuration Parameters dialog box.  When the ExecTime reaches StopTime, the application stops running.	Yes
ViewMode	Display either all scopes or a single scope on the target computer. Value is 'all' or a single scope index. This property is active only if the environment property TargetScope is set to enabled.	Yes

## xpctarget.xpc.set (target application object)

---

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the value of the properties after the indicated settings have been made.

### Examples

Get a list of writable properties for a scope object.

```
set(tg)
ans =
    StopTime: {}
    SampleTime: {}
    ViewMode: {}
    LogMode: {}
    ShowParameters: {}
    ShowSignals: {}
```

Change the property `ShowSignals` to `on`.

```
tg.set('showsignals', 'on') or set(tg, 'showsignals', 'on')
```

As an alternative to the method `set`, use the target object property `ShowSignals`. In the MATLAB window, type

```
tg.showsignals = 'on'
```

### See Also

```
get | set | xpctarget.xpc.get (target application object) |
xpctarget.xpcsc.get (scope object) | xpctarget.xpcsc.set
(scope object)
```

### How To

- “Application and Driver Scripts”



**Purpose** Change writable target object parameters

**Syntax** `setparam(target_object, parameter_index, parameter_value)`

**Arguments**

<code>target_object</code>	Name of an existing target object. The default name is <code>tg</code> .
<code>parameter_index</code>	Index number of the parameter.
<code>parameter_value</code>	Value for a target object parameter.

**Description** Method of a target object. Set the value of the target parameter. This method returns a structure that stores the parameter index, previous parameter values, and new parameter values in the following fields:

- `parIndexVec`
- `OldValues`
- `NewValues`

**Examples** Set the value of parameter index 5 to 100.

```
setparam(tg, 5, 100)
ans =
parIndexVec: 5
OldValues: 400
NewValues: 100
```

Simultaneously set values for multiple parameters. Use the cell array format to specify new parameter values.

```
setparam(tg, [1 5], {10,100})
ans =
parIndexVec: [1 5]
OldValues: {[2] [4]}
NewValues: {[10] [100]}
```

# xpctarget.xpc.start (target application object)

---

**Purpose** Start execution of target application on target computer

**Syntax** MATLAB command line

```
start(target_object)
target_object.start
+target_object
```

**Target computer command line**

```
start
```

**Arguments** target\_object Name of a target object. The default name is tg.

**Description** Method of both target and scope objects. Starts execution of the target application represented by the target object. Before using this method, the target application must be created and loaded on the target computer. If a target application is running, this command has no effect.

**Examples** Start the target application represented by the target object tg.

```
+tg
tg.start
start(tg)
```

**See Also** xpctarget.xpc.stop (target application object)  
| xpctarget.xpc.load | xpctarget.xpc.unload |  
xpctarget.xpcsc.stop (scope object)

# xpctarget.xpc.stop (target application object)

---

**Purpose** Stop execution of target application on target computer

**Syntax** MATLAB command line

```
stop(target_object)  
target_object.stop  
-target_object
```

**Target computer command line**

```
stop
```

**Arguments** target\_object Name of a target object.

**Description** Stops execution of the target application represented by the target object. If the target application is stopped, this command has no effect.

**Examples** Stop the target application represented by the target object tg.

```
stop(tg) or tg.stop or -tg
```

**See Also** xpctarget.xpc.start (target application object) | xpctarget.xpcsc.stop (scope object) | xpctarget.xpcsc.start (scope object)

# xpctarget.xpc.targetping

---

**Purpose** Test communication between host and target computers

**Syntax** targetping(target\_object)  
target\_object.targetping

**Arguments** target\_object Name of a target object.

**Description** Method of a target object. Use this method to ping a target computer from the host computer. This method returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target, otherwise it returns `failed`.

This function works with both RS-232 and TCP/IP communication.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**Examples** Ping the communication between the host and the target object tg.  
`targetping(tg)` or `tg.targetping`

**See Also** xpctarget.xpc

**Purpose** Remove current target application from target computer

**Syntax** unload(target\_object)  
target\_object.unload

**Arguments** target\_object Name of a target object that represents a target application.

**Description** Method of a target object. The kernel goes into loader mode and is ready to download new target application from the host computer.

---

**Note** If you are running in StandAlone mode, this command has no effect. To unload and reload a new application, you must rebuild the standalone application with the new application, then reboot the target computer with the updated standalone application.

---

**Examples** Unload the target application represented by the target object tg.

unload(tg) or tg.unload

**See Also** xpctarget.xpc.load | xpctarget.xpc.reboot

# xpctarget.xpcfs Class

**Purpose** Control and access properties of file scopes

**Description** The scope gets a data package from the kernel and stores the data in a file in the target computer file system. Depending on the setting of WriteMode, the file size is or is not continuously updated. You can then transfer the data to another computer for examination or plotting.

## Methods

These methods are inherited from xpctarget.xpcsc Class.

Method	Description
xpctarget.xpcsc.addsignals	Add signals to scope represented by scope object
xpctarget.xpcsc.get(scope object)	Return property values for scope objects
xpctarget.xpcsc.removesignals	Remove signals from scope represented by scope object
xpctarget.xpcsc.set(scope object)	Change property values for scope objects
xpctarget.xpcsc.start(scope object)	Start execution of scope on target computer
xpctarget.xpcsc.stop(scope object)	Stop execution of scope on target computer
xpctarget.xpcsc.trigger	Software-trigger start of data acquisition for scope(s)

## Properties

These properties are inherited from xpctarget.xpcsc Class.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

## xpctarget.xpcf Class

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes



Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcfs.

Property	Description	Writeable
AutoRestart	<p>Values are 'on' and 'off'.</p> <p>For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b>, then stop.</p> <p>If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.</p> <p>To use the DynamicFileName property, set AutoRestart to 'on' first.</p>	No

# xpctarget.xpcf Class

Property	Description	Writeable
	<p>For host or target scopes, this parameter has no effect.</p>	
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use DynamicFileName, set AutoRestart to 'on' first. When you enable DynamicFileName, configure Filename to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.</p> <p>You can enable the creation of up to 99999999 files (&lt;%%%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

Property	Description	Writeable
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to <code>FileName</code>. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, <code>ScopeId</code>, and the beginning letters of the first signal added to the scope.</p> <p>If you set <code>DynamicFileName</code> and <code>AutoRestart</code> to 'on', configure <code>Filename</code> to dynamically increment. Use a base file name, an underscore (<code>_</code>), and a <code>&lt; &gt;</code> specifier. Within the specifier, enter one to eight <code>%</code> symbols. Each symbol <code>%</code> represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for <code>Filename</code>, <code>C:\work\file_&lt;%%&gt;.dat</code> creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre>	No

# xpctarget.xpcfs Class

Property	Description	Writeable
	<p>The last file name of this series will be <code>file_999.dat</code>. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	
<code>MaxWriteFileSize</code>	<p>Provide the maximum size of <code>Filename</code>, in bytes. This value must be a multiple of <code>WriteSize</code>. Default is <code>536870912</code>.</p> <p>When the size of a log file reaches <code>MaxWriteFileSize</code>, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

Property	Description	Writeable
Mode	<p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"> <li>• For target scopes, use <code>DisplayMode</code>.</li> <li>• For file scopes, use <code>WriteMode</code>.</li> <li>• For host scopes, this parameter has no effect.</li> </ul>	Yes
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
WriteSize	Enter the block size, in bytes, of the data chunks. This parameter	Yes

## xpctarget.xpcfs Class

---

Property	Description	Writeable
	<p>specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	

**Purpose** Add signals to scope represented by scope object

**Syntax** MATLAB command line

```
addsignal(scope_object_vector, signal_index_vector)
scope_object_vector.addsignal(signal_index_vector)
```

**Target command line**

```
addsignal scope_index = signal_index, signal_index, . . .
```

## Arguments

scope_object_vector	Name of a single scope object or the name of a vector of scope objects.
signal_index_vector	For one signal, use a single number. For two or more signals, enclose numbers in brackets and separate with commas.
scope_index	Single scope index.

## Description

`addsignal` adds signals to a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_object_vector` has two or more scope objects, the same signals are assigned to each scope.

---

**Note** You must stop the scope before you can add a signal to it.

---

## Examples

Add signals 0 and 1 from the target object `tg` to the scope object `sc1`. The signals are added to the scope, and the scope object property `Signals` is updated to include the added signals.

```
sc1 = getscope(tg,1)
addsignal(sc1,[0,1]) or sc1.addsignal([0,1])
```

## xpctarget.xpcsc.addsignal

---

Display a list of properties and values for the scope object `sc1` with the property `Signals`, as shown below.

```
sc1.Signals
Signals          = 1 : Signal Generator
                  0 : Integrator1
```

Another way to add signals without using the method `addsignal` is to use the scope object method `set`.

```
set(sc1,'Signals', [0,1]) or sc1.set('signals',[0,1])
```

Or, to directly assign signal values to the scope object property `Signals`,

```
sc1.signals = [0,1]
```

### See Also

`xpctarget.xpcsc.remsignal` | `xpctarget.xpcsc.set` (scope object) | `xpctarget.xpc.addscope` | `xpctarget.xpc.getsignalid`



# xpctarget.xpcsc.get (scope object)

**Purpose** Return property values for scope objects

**Syntax**  
`get(scope_object_vector)`  
`get(scope_object_vector, 'scope_object_property')`  
`get(scope_object_vector, scope_object_property_vector)`

**Arguments**

<code>target_object</code>	Name of a target object.
<code>scope_object_vector</code>	Name of a single scope or name of a vector of scope objects.
<code>scope_object_property</code>	Name of a scope object property.

**Description** `get` gets the value of readable scope object properties from a scope object or the same property from each scope object in a vector of scope objects. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
AutoRestart	Values are 'on' and 'off'.  For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b> , then stop.	No

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
	<p>If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.</p> <p>For host or target scopes, this parameter has no effect.</p> <p>To use the <code>DynamicFileName</code> property, set <code>AutoRestart</code> to 'on' first.</p>	
Data	<p>Contains the output data for a single data package from a scope.</p> <p>For target or file scopes, this parameter has no effect.</p>	No
Decimation	<p>A number <math>n</math>, where every <math>n</math>th sample is acquired in a scope window.</p>	Yes
DisplayMode	<p>For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For host or file scopes, this parameter has no effect.</p> <p>.</p>	Yes
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use <code>DynamicFileName</code>, set <code>AutoRestart</code> to 'on' first. When you enable <code>DynamicFileName</code>, configure <code>Filename</code> to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.</p>	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
	<p>You can enable the creation of up to 99999999 files (&lt;%%%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or file scopes, this parameter has no effect.</p>	
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to <code>FileName</code>. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, <code>ScopeId</code>, and the beginning letters of the first signal added to the scope.</p> <p>If you set <code>DynamicFileName</code> and <code>AutoRestart</code> to 'on', configure <code>Filename</code> to dynamically increment. Use a base file name, an underscore (<code>_</code>), and a <code>&lt; &gt;</code> specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name. The specifier can appear anywhere in the file name. For example, the following value for <code>Filename</code>, C:\work\file_&lt;%%&gt;.dat creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre> <p>The last file name of this series will be <code>file_999.dat</code>. If the function is still logging data when the last file name reaches its maximum</p>	No

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
	<p>size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	
MaxWriteFileSize	<p>Provide the maximum size of <code>Filename</code>, in bytes. This value must be a multiple of <code>WriteSize</code>. Default is 536870912.</p> <p>When the size of a log file reaches <code>MaxWriteFileSize</code>, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.</p>	Yes
Grid	<p>Values are 'on' and 'off'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes
Mode	<hr/> <p><b>Note</b> The <code>Mode</code> property will be removed in a future release.</p> <ul style="list-style-type: none"><li>• For target scopes, use <code>DisplayMode</code>.</li><li>• For file scopes, use <code>WriteMode</code>.</li><li>• For host scopes, this parameter has no effect.</li></ul> <hr/>	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
Time	Contains the time data for a single data package from a scope.	No
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes

## xpctarget.xpcsc.get (scope object)

Property	Description	Writable
WriteSize	<p>Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
YLimit	<p>Minimum and maximum y-axis values. This property can be set to 'auto'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

### Examples

List the readable properties, along with their current values. This is given in the form of a structure whose field names are the property names and whose field values are property values.

```
get(sc)
```

List the value for the scope object property Type. Notice that the property name is a string, in quotation marks, and is not case sensitive.

```
get(sc, 'type')
ans = Target
```

### See Also

```
get | set | xpctarget.xpcsc.set (scope object) |
xpctarget.xpc.set (target application object)
```



**Purpose** Base class for the scope classes

**Description** This is the base class for the scope classes, `xpctarget.xpcfs Class`, `xpctarget.xpcshost Class`, and `xpctarget.xpcscctg Class`. All methods and properties are inherited by the derived classes. When a mixture of derived classes are stored in a scope collection, only the base class methods and properties are available. The scope class constructors are `Private` and are not intended to be called from the MATLAB prompt.

A scope acquires data from the target application and displays that data on the target computer, uploads the data to the host computer, or stores that data in a file in the target computer file system. The target, host, or file scopes run on the target computer.

### Methods

These methods are inherited by the derived classes.

Method	Description
<code>xpctarget.xpcsc.addsignals</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get</code> (scope object)	Return property values for scope objects
<code>xpctarget.xpcsc.removesignals</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set</code> (scope object)	Change property values for scope objects
<code>xpctarget.xpcsc.start</code> (scope object)	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop</code> (scope object)	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software trigger start of data acquisition for scope(s)

### Properties

These properties are inherited by the derived classes.

## xpctarget.xpcsc Class

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number n, where every nth sample is acquired in a scope window.	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <code>TriggerMode</code> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes

Property	Description	Writable
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes

## xpctarget.xpcsc Class

---

<b>Property</b>	<b>Description</b>	<b>Writable</b>
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

**Purpose** Remove signals from scope represented by scope object

**Syntax** **MATLAB command line**

```
remsignal(scope_object)
remsignal(scope_object, signal_index_vector)
scope_object.remsignal(signal_index_vector)
```

**Target command line**

```
remsignal scope_index = signal_index, signal_index, . . .
```

**Arguments**

scope_object	MATLAB object created with the target object method <code>addscope</code> or <code>getscope</code> .
signal_index_vector	Index numbers from the scope object property <code>Signals</code> . This argument is optional, and if it is left out all signals are removed.
signal_index	Single signal index.

**Description** `remsignal` removes signals from a scope object. The signals must be specified by their indices, which you can retrieve using the target object method `getsignalid`. If the `scope_index_vector` has two or more scope objects, the same signals are removed from each scope. The argument `signal_index` is optional; if it is left out, all signals are removed.

---

**Note** You must stop the scope before you can remove a signal from it.

---

**Examples** Remove signals 0 and 1 from the scope represented by the scope object `sc1`.

```
sc1.get('signals')
ans= 0 1
```

# xpctarget.xpcsc.remsignal

---

Remove signals from the scope on the target computer with the scope object property `Signals` updated.

```
remsignal(sc1,[0,1])
```

or

```
sc1.remsignal([0,1])
```

## See Also

[xpctarget.xpcsc.remsignal](#) | [xpctarget.xpc.getsignalid](#)

## Purpose

Change property values for scope objects

## Syntax

```
set(scope_object_vector)
set(scope_object_vector, property_name1, property_value1,
property_name2, property_value2, . . .)
scope_object_vector.set('property_name1', property_value1,
. . .)
set(scope_object, 'property_name', property_value, . . .)
```

## Arguments

`scope_object` Name of a scope object or a vector of scope objects.

`'property_name'` Name of a scope object property. Always use quotation marks.

`property_value` Value for a scope object property. Always use quotation marks for character strings; quotation marks are optional for numbers.

## Description

Method for scope objects. Sets the properties of the scope object. Not all properties are user writable. Scope object properties let you select signals to acquire, set triggering modes, and access signal information from the target application. You can view and change these properties using scope object methods.

Properties must be entered in pairs or, using the alternate syntax, as one-dimensional cell arrays of the same size. This means they must both be row vectors or both column vectors, and the corresponding values for properties in `property_name_vector` are stored in `property_value_vector`.

The function `set` typically does not return a value. However, if called with an explicit return argument, for example, `a = set(target_object, property_name, property_value)`, it returns the values of the properties after the indicated settings have been made.

The properties for a scope object are listed in the following table. This table includes descriptions of the properties and the properties you can change directly by assigning a value.

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
AutoRestart	<p>Values are 'on' and 'off'.</p> <p>For file scopes, enable the file scope to collect data up to the number of samples (NumSamples), then start over again, appending the new data to the end of the signal data file. Clear the <b>AutoRestart</b> check box to have the file scope collect data up to <b>Number of samples</b>, then stop.</p> <p>If the named signal data file already exists when you start the target application, the software overwrites the old data with the new signal data.</p> <p>For host or target scopes, this parameter has no effect.</p> <p>To use the DynamicFileName property, set AutoRestart to 'on' first.</p>	No
Data	<p>Contains the output data for a single data package from a scope.</p> <p>For target or file scopes, this parameter has no effect.</p>	No
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes
DisplayMode	<p>For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.</p> <p>For host or file scopes, this parameter has no effect.</p> <p>.</p>	Yes



## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
DynamicFileName	<p>Values are 'on' and 'off'. By default, the value is 'off'.</p> <p>Enable the ability to dynamically create multiple log files for file scopes.</p> <p>To use DynamicFileName, set AutoRestart to 'on' first. When you enable DynamicFileName, configure Filename to create incrementally numbered file names for the multiple log files. Failure to do so causes an error when you try to start the scope.</p> <p>You can enable the creation of up to 99999999 files (&lt;%%%%%%&gt;.dat). The length of a file name, including the specifier, cannot exceed eight characters.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes
Filename	<p>Provide a name for the file to contain the signal data. By default, the target computer writes the signal data to a file named C:\data.dat for scope blocks. Note that for file scopes created through the MATLAB interface, no name is initially assigned to FileName. After you start the scope, the software assigns a name for the file to acquire the signal data. This name typically consists of the scope object name, ScopeId, and the beginning letters of the first signal added to the scope.</p> <p>If you set DynamicFileName and AutoRestart to 'on', configure Filename to dynamically increment. Use a base file name, an underscore (_), and a &lt; &gt; specifier. Within the specifier, enter one to eight % symbols. Each symbol % represents a decimal location in the file name.</p>	No

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
	<p>The specifier can appear anywhere in the file name. For example, the following value for <code>Filename</code>, <code>C:\work\file_&lt;%%&gt;.dat</code> creates file names with the following pattern:</p> <pre>file_001.dat file_002.dat file_003.dat</pre> <p>The last file name of this series will be <code>file_999.dat</code>. If the function is still logging data when the last file name reaches its maximum size, the function starts from the beginning and overwrites the first file name in the series. If you do not retrieve the data from existing files before they are overwritten, the data is lost.</p> <p>For host or target scopes, this parameter has no effect.</p>	
<code>MaxWriteFileSize</code>	<p>Provide the maximum size of <code>Filename</code>, in bytes. This value must be a multiple of <code>WriteSize</code>. Default is 536870912.</p> <p>When the size of a log file reaches <code>MaxWriteFileSize</code>, the software creates a subsequently numbered file name, and continues logging data to that file, up until the highest log file number you have specified. If the software cannot create additional log files, it overwrites the first log file.</p>	Yes
<code>Grid</code>	<p>Values are 'on' and 'off'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
Mode	<p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"><li>• For target scopes, use DisplayMode.</li><li>• For file scopes, use WriteMode.</li><li>• For host scopes, this parameter has no effect.</li></ul>	Yes
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set TriggerMode to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No
Time	Contains the time data for a single data package from a scope.	No
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is</p>	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
	acquired one sample after the last sample of the triggering scope.	
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes
WriteMode	<p>For file scopes, specify when a file allocation table (FAT) entry is updated. Values are 'Lazy' or 'Commit'. Both modes write the signal data to the file. With 'Commit' mode, each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size. With 'Lazy' mode, the FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not know the actual file size (the file contents, however, will be intact).</p> <p>For host or target scopes, this parameter has no effect.</p> <p>.</p>	Yes

## xpctarget.xpcsc.set (scope object)

Property	Description	Writable
WriteSize	<p>Enter the block size, in bytes, of the data chunks. This parameter specifies that a memory buffer, of length number of samples (NumSamples), collect data in multiples of WriteSize. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance.</p> <p>If you experience a system crash, you can expect to lose an amount of data the size of WriteSize.</p> <p>For host or target scopes, this parameter has no effect.</p>	Yes
YLimit	<p>Minimum and maximum y-axis values. This property can be set to 'auto'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

### Examples

Get a list of writable properties for a scope object.

```
sc1 = getscope(tg,1)
set(sc1)
ans=
    NumSamples: {}
    Decimation: {}
    TriggerMode: {5x1 cell}
    TriggerSignal: {}
    TriggerLevel: {}
    TriggerSlope: {4x1 cell}
    TriggerScope: {}
    TriggerSample: {}
    Signals: {}
    NumPrePostSamples: {}
    Mode: {5x1 cell}
    YLimit: {}
```

Grid: {}

The property value for the scope object sc1 is changed to on:

```
sc1.set('grid', 'on') or set(sc1, 'grid', 'on')
```

### **See Also**

```
get | set | xpctarget.xpcsc.get (scope object) |  
xpctarget.xpc.set (target application object) |  
xpctarget.xpc.get (target application object)
```

# xpctarget.xpcsc.start (scope object)

---

**Purpose** Start execution of scope on target computer

**Syntax** MATLAB command line

```
start(scope_object_vector)
scope_object_vector.start
+scope_object_vector
start(getscope((target_object, signal_index_vector)))
```

**Target computer command line**

```
startscope scope_index
startscope 'all'
```

## Arguments

<code>target_object</code>	Name of a target object.
<code>scope_object_vector</code>	Name of a single scope object, name of vector of scope objects, list of scope object names in vector form [ <code>scope_object1</code> , <code>scope_object2</code> ], or the target object method <code>getscope</code> , which returns a <code>scope_object</code> vector.
<code>signal_index_vector</code>	Index for a single scope or list of scope indices in vector form.
<code>scope_index</code>	Single scope index.

## Description

Method for a scope object. Starts a scope on the target computer represented by a scope object on the host computer. This method might not start data acquisition, which depends on the trigger settings. Before using this method, you must create a scope. To create a scope, use the target object method `addscope` or add xPC Target scope blocks to your Simulink model.



## Examples

Start one scope with the scope object sc1.

```
sc1 = getscope(tg,1) or sc1 = tg.getscope(1)
start(sc1) or sc1.start or +sc1
```

or type

```
start(getscope(tg,1))
```

Start two scopes.

```
somescopes = getscope(tg,[1,2]) or somescopes =
tg.getscope([1,2])
start(somescopes) or somescopes.start
```

or type

```
sc1 = getscope(tg,1) or sc1 =tg.getscope(1)
sc2 = getscope(tg,2) or sc2 = tg.getscope(2)
start([sc1,sc2])
```

or type

```
start(getscope(tg,[1,2]))
```

Start all scopes:

```
allscopes = getscope(tg) or allscopes = tg.getscope
start(allscopes) or allscopes.start or +allscopes
```

or type

```
start(getscope(tg)) or start(tg.getscope)
```

## See Also

xpctarget.xpc.getscope | xpctarget.xpc.stop (target application object) | xpctarget.xpcsc.stop (scope object)

# xpctarget.xpcsc.stop (scope object)

---

**Purpose** Stop execution of scope on target computer

**Syntax** MATLAB command line

```
stop(scope_object_vector)
scope_object.stop
-scope_object
stop(getscope(target_object, signal_index_vector))
```

**Target computer command line**

```
stopscope scope_index
stopscope 'all'
```

## Arguments

target_object	Name of a target object.
scope_object_vector	Name of a single scope object, name of vector of scope objects, list of scope object names in a vector form [ <code>scope_object1</code> , <code>scope_object2</code> ], or the target object method <code>getscope</code> , which returns a <code>scope_object</code> vector.
signal_index_vector	Index for a single scope or list of scope indices in vector form.
scope_index	Single scope index.

**Description** Method for scope objects. Stops the scopes represented by the scope objects.

## Examples

Stop one scope represented by the scope object `sc1`.

```
stop(sc1) or sc1.stop or -sc1
```

Stop all scopes with a scope object vector `allscopes` created with the command

## xpctarget.xpcsc.stop (scope object)

---

```
allscopes = getscope(tg) or allscopes = tg.getscope.  
stop(allscopes) or allscopes.stop or -allscopes
```

or type

```
stop(getscope(tg)) or stop(tg.getscope)
```

### **See Also**

```
xpctarget.xpc.getscope | xpctarget.xpc.stop (target  
application object) | xpctarget.xpc.start (target application  
object) | xpctarget.xpcsc.start (scope object)
```

# xpctarget.xpcsc.trigger

---

<b>Purpose</b>	Software-trigger start of data acquisition for scope(s)
<b>Syntax</b>	<code>trigger(scope_object_vector)</code> or <code>scope_object_vector.trigger</code>
<b>Arguments</b>	<code>scope_object_vector</code> Name of a single scope object, name of a vector of scope objects, list of scope object names in a vector form [ <code>scope_object1</code> , <code>scope_object2</code> ], or the target object method <code>getscope</code> , which returns a <code>scope_object</code> vector.
<b>Description</b>	<p>Method for a scope object. If the scope object property <code>TriggerMode</code> has a value of 'software', this function triggers the scope represented by the scope object to acquire the number of data points in the scope object property <code>NumSamples</code>.</p> <p>Note that only scopes with type <code>host</code> store data in the properties <code>scope_object.Time</code> and <code>scope_object.Data</code>.</p>
<b>Examples</b>	<p>Set a single scope to software trigger, trigger the acquisition of one set of samples, and plot data.</p> <pre>sc1 = tg.addscope('host',1) or sc1=addscope(tg,'host',1) sc1.triggermode = 'software' tg.start, or start(tg), or +tg sc1.start or start(sc1) or +sc1 sc1.trigger or trigger(sc1) plot(sc1.time, sc1.data) sc1.stop or stop(sc1) or -sc1 tg.stop or stop(tg) or -tg1</pre> <p>Set all scopes to software trigger and trigger to start.</p> <pre>allscopes = tg.getscopes allscopes.triggermode = 'software' allscopes.start or start(allscopes) or +allscopes</pre>

allscopes.trigger or trigger(allscopes)

# xpctarget.xpcschoost Class

**Purpose** Control and access properties of host scopes

**Description** The scope gets a data package from the kernel, waits for an upload command from the host computer, and uploads the data to the host. The host computer displays the data using a scope viewer or other MATLAB functions.

## Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
<code>xpctarget.xpcsc.addsignals</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get(scope object)</code>	Return property values for scope objects
<code>xpctarget.xpcsc.removesignals</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set(scope object)</code>	Change property values for scope objects
<code>xpctarget.xpcsc.start(scope object)</code>	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop(scope object)</code>	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software-trigger start of data acquisition for scope(s)

## Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

## xpctarget.xpcschoost Class

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes



Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcschoost.

Property	Description	Writeable
Data	Contains the output data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No
Time	Contains the time data for a single data package from a scope. For target or file scopes, this parameter has no effect.	No

# xpctarget.xpcsc Class

**Purpose** Control and access properties of target scopes

**Description** The kernel acquires a data package and the scope displays the data on the target computer screen. Depending on the setting of `DisplayMode`, the data may be displayed numerically or graphically by a redrawing, sliding, and rolling display.

## Methods

These methods are inherited from `xpctarget.xpcsc Class`.

Method	Description
<code>xpctarget.xpcsc.addsignals</code>	Add signals to scope represented by scope object
<code>xpctarget.xpcsc.get(scope object)</code>	Return property values for scope objects
<code>xpctarget.xpcsc.removesignals</code>	Remove signals from scope represented by scope object
<code>xpctarget.xpcsc.set(scope object)</code>	Change property values for scope objects
<code>xpctarget.xpcsc.start(scope object)</code>	Start execution of scope on target computer
<code>xpctarget.xpcsc.stop(scope object)</code>	Stop execution of scope on target computer
<code>xpctarget.xpcsc.trigger</code>	Software-trigger start of data acquisition for scope(s)

## Properties

These properties are inherited from `xpctarget.xpcsc Class`.

Property	Description	Writable
Application	Name of the Simulink model associated with this scope object.	No
Decimation	A number $n$ , where every $n$ th sample is acquired in a scope window.	Yes

Property	Description	Writable
NumPrePostSamples	Number of samples collected before or after a trigger event. The default value is 0. Entering a negative value collects samples before the trigger event. Entering a positive value collects samples after the trigger event. If you set <b>TriggerMode</b> to 'FreeRun', this property has no effect on data acquisition.	Yes
NumSamples	<p>Number of contiguous samples captured during the acquisition of a data package. If the scope stops before capturing this number of samples, the scope has the collected data up to the end of data collection, then has zeroes for the remaining uncollected data. Note that you should know what type of data you are collecting, it is possible that your data contains zeroes.</p> <p>For file scopes, this parameter works in conjunction with the <b>AutoRestart</b> check box. If the <b>AutoRestart</b> box is selected, the file scope collects data up to <b>Number of Samples</b>, then starts over again, overwriting the buffer. If the <b>AutoRestart</b> box is not selected, the file scope collects data only up to <b>Number of Samples</b>, then stops.</p>	Yes
ScopeId	A numeric index, unique for each scope.	No
Signals	List of signal indices from the target object to display on the scope.	Yes
Status	Indicate whether data is being acquired, the scope is waiting for a trigger, the scope has been stopped (interrupted), or acquisition is finished. Values are 'Acquiring', 'Ready for being Triggered', 'Interrupted', and 'Finished'.	No

## xpctarget.xpcstg Class

Property	Description	Writable
TriggerLevel	If TriggerMode is 'Signal', indicates the value the signal has to cross to trigger the scope and start acquiring data. The trigger level can be crossed with either a rising or falling signal.	Yes
TriggerMode	Trigger mode for a scope. Valid values are 'FreeRun' (default), 'Software', 'Signal', and 'Scope'.	Yes
TriggerSample	<p>If TriggerMode is 'Scope', then TriggerSample specifies which sample of the triggering scope the current scope should trigger on. For example, if TriggerSample is 0 (default), the current scope triggers on sample 0 (first sample acquired) of the triggering scope. This means that the two scopes will be perfectly synchronized. If TriggerSample is 1, the first sample (sample 0) of the current scope will be at the same instant as sample number 1 (second sample in the acquisition cycle) of the triggering scope.</p> <p>As a special case, setting TriggerSample to -1 means that the current scope is triggered at the end of the acquisition cycle of the triggering scope. Thus, the first sample of the triggering scope is acquired one sample after the last sample of the triggering scope.</p>	Yes
TriggerScope	If TriggerMode is 'Scope', identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered. You do this by setting the slave scope property TriggerScope to the scope index of the master scope.	Yes

Property	Description	Writable
TriggerSignal	If TriggerMode is 'Signal', identifies the block output signal to use for triggering the scope. You identify the signal with a signal index from the target object property Signal.	Yes
TriggerSlope	If TriggerMode is 'Signal', indicates whether the trigger is on a rising or falling signal. Values are 'Either' (default), 'Rising', and 'Falling'.	Yes
Type	Determines whether the scope is displayed on the host computer or on the target computer. Values are 'Host', 'Target', and 'File'.	Yes

These properties are specific to class xpcsctg.

Property	Description	Writeable
DisplayMode	For target scopes, indicate how a scope displays the signals. Values are 'Numerical', 'Redraw' (default), 'Sliding', and 'Rolling'.  For host or file scopes, this parameter has no effect.	Yes
Grid	Values are 'on' and 'off'.  For host or file scopes, this parameter has no effect.	Yes

# xpctarget.xpcsctg Class

Property	Description	Writeable
Mode	<p><b>Note</b> The Mode property will be removed in a future release.</p> <ul style="list-style-type: none"><li>• For target scopes, use DisplayMode.</li><li>• For file scopes, use WriteMode.</li><li>• For host scopes, this parameter has no effect.</li></ul>	Yes
YLimit	<p>Minimum and maximum y-axis values. This property can be set to 'auto'.</p> <p>For host or file scopes, this parameter has no effect.</p>	Yes

**Purpose** Test communication between host and target computers

**Syntax**  
xpctargetping  
xpctargetping('mode', 'arg1', 'arg2')

**Arguments**

mode	Optionally, enter the communication mode:
------	---

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

TCP/IP Enable TCP/IP connection with target computer.

RS232 Enable RS-232 connection with target computer.

arg1	Optionally, enter an argument based on the mode value:
------	--

IP If mode is 'TCP/IP', enter the IP address of the target computer.

COM If mode is 'RS232', enter the host COM port.

arg2	Optionally, enter an argument based on the mode value:
------	--

Port If mode is 'TCP/IP', enter the port number for the target computer.

Baud rate If mode is 'RS232', enter the baud rate for the connection between the host and target computer.

# xpctargetping

---

## Description

Pings the target computer from the host computer and returns either `success` or `failed`. If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
xpctargetping
```

If you have multiple target computers in your system, use the following syntax to identify the target computer to ping.

```
xpctargetping('mode', 'arg1', 'arg2')
```

This function returns `success` if the xPC Target kernel is loaded and running and communication is working between host and target.

This function works with both RS-232 and TCP/IP communication.

## Examples

Check for communication between the host computer and target computer.

```
xpctargetping
```

If you have a serial connection with the target computer you want to check, use the following syntax.

```
xpctargetping('RS232', 'COM1', '115200')
```

## How To

- “Run Confidence Test on Configuration”



**Purpose** Open Real-Time xPC Target Spy window on host computer

**Syntax**

```
xpctargetspy
xpctargetspy(target_object)
xpctargetspy('target_object_name')
```

**Arguments**

target_object	Variable name to reference the target object.
target_object_name	Target object name as specified in the xPC Target Explorer.

**Description** This graphical user interface (GUI) allows you to upload displayed data from the target computer. By default, xpctargetspy opens a Real-Time xPC Target Spy window for the target object, tg. If you have multiple target computers in your system, you can call the xpctargetspy function for a particular target object, target\_object.

If you have one target computer, or if you designate a target computer as the default one in your system, use the syntax

```
xpctargetspy
```

If you have multiple target computers in your system, use xpctarget.xpc to create the additional target object first.

```
target_object=xpctarget.xpc('mode', 'arg1', 'arg2')
```

Then, use the following syntax.

```
xpctargetspy(target_object)
```

If you have a target computer object in the xPC Target Explorer, you can use the following syntax.

```
target_object=xpctarget.xpc('target_object_name')
```

The behavior of this function depends on the value for the environment property TargetScope:

- If TargetScope is enabled, a single graphics screen is uploaded. The screen is not continually updated because of a higher data volume when a target graphics card is in VGA mode. You must explicitly request an update. To manually update the host screen with another target screen, move the pointer into the Real-Time xPC Target Spy window and right-click to select **Update xPC Target Spy**.
- If TargetScope is disabled, text output is transferred once every second to the host and displayed in the window.

### Examples

To open the Real-Time xPC Target Spy window for a default target computer, `tg`, in the MATLAB window, type

```
xpctargetspy
```

To open the Real-Time xPC Target Spy window for a target computer, `tg1`, in the MATLAB window, type

```
xpctargetspy(tg1)
```

If you have multiple target computers in your system, use `xpctarget.xpc` to create the additional target object, `tg2`, first.

```
tg2=xpctarget.xpc('RS232', 'COM1', '115200')
```

Then, use the following syntax.

```
xpctargetspy(tg2)
```

**Purpose** Test xPC Target installation

**Syntax**

```
xpctest
xpctest('noreboot')
xpctest('-noreboot')
xpctest('target_name')
xpctest('target_name','noreboot')
xpctest('target_name','-noreboot')
```

**Arguments**

'target_name'	Name of target computer to test.
'noreboot'	Only one possible option. Skips the reboot test. Use this option if the target hardware does not support software rebooting. Value is 'noreboot' or '-noreboot'.

**Description** xpctest is a series of xPC Target tests to check the functioning of the following xPC Target tasks:

- Initiate communication between the host and target computers.
- Reboot the target computer to reset the target environment.
- Build a target application on the host computer.
- Download a target application to the target computer.
- Check communication between the host and target computers using commands.
- Execute a target application.
- Compare the results of a simulation and the target application run.

xpctest('noreboot') or xpctest('-noreboot') skips the reboot test on the default target computer. Use this option if target hardware does not support software rebooting.

xpctest('target\_name') runs the tests on the target computer identified by 'target\_name'.

`xpctest('target_name', 'noreboot')` or  
`xpctest('target_name', '-noreboot')` runs the tests on the target computer identified by 'target\_name', but skips the reboot test.

## Examples

If the target hardware does not support software rebooting, or to skip the reboot test, in the MATLAB window, type

```
xpctest(' -noreboot')
```

To run `xpctest` on a specified target computer, for example TargetPC1, type

```
xpctest('TargetPC1')
```

## How To

- “Run Confidence Test on Configuration”
- “Test 1: Ping Using System Ping”

**Purpose** Disconnect target computer from current client application

**Syntax** `xpcwwenable`  
`xpcwwenable('target_obj_name')`

**Description** `xpcwwenable` disconnects the target application from the MATLAB interface so you can connect to the Web browser. You can also use this function to connect to the MATLAB interface after using a Web browser, or to switch to another Web browser.

`xpcwwenable('target_obj_name')` disconnects the target application on `target_obj_name` (for example 'TargetPC1') from the MATLAB interface.



# xPC Target API Reference for C

---

- “C API Functions” on page 2-2
- “C API Error Messages” on page 2-10
- “C API Structures and Functions — Alphabetical List” on page 2-14

## C API Functions

### In this section...

“Target Computers” on page 2-2  
“Target Applications” on page 2-3  
“Scopes” on page 2-4  
“Parameters” on page 2-6  
“Signals” on page 2-7  
“Data Logs” on page 2-7  
“File Systems” on page 2-8  
“Errors” on page 2-9

### Target Computers

xPCCloseConnection	Close RS-232 or TCP/IP communication connection
xPCClosePort	Close RS-232 or TCP/IP communication connection
xPCDeRegisterTarget	Delete target communication properties from xPC Target API library
xPCFreeAPI	Unload xPC Target DLL
xPCGetEcho	Return display mode for target message window
xPCGetLoadTimeOut	Return timeout value for communication between host computer and target computer
xPCInitAPI	Initialize xPC Target DLL
xPCIsAppRunning	Return target application running status
xPCOpenConnection	Open connection to target computer



xPCOpenSerialPort	Open RS-232 connection to xPC Target system
xPCOpenTcpIpPort	Open TCP/IP connection to xPC Target system
xPCReboot	Reboot target computer
xPCRegisterTarget	Register target with xPC Target API library
xPCReOpenPort	Reopen communication channel
xPCSetEcho	Turn message display on or off
xPCSetLoadTimeOut	Change initialization timeout value between host computer and target computer
xPCTargetPing	Ping target computer
xPCUnloadApp	Unload target application

## Target Applications

xPCAverageTET	Return average task execution time
xPCGetAPIVersion	Get version number of xPC Target API
xPCGetAppName	Return target application name
xPCGetExecTime	Return target application execution time
xPCGetSampleTime	Return target application sample time
xPCGetSessionTime	Return length of time xPC Target kernel has been running
xPCGetStopTime	Return stop time
xPCGetTargetVersion	Get xPC Target kernel version
xPCIsOverloaded	Return target computer overload status

xPCLoadParamSet	Restore parameter values
xPCMaximumTET	Copy maximum task execution time to array
xPCMinimumTET	Copy minimum task execution time to array
xPCSaveParamSet	Save parameter values of target application
xPCSetSampleTime	Change target application sample time
xPCSetStopTime	Change target application stop time
xPCStartApp	Start target application
xPCStopApp	Stop target application

## Scopes

scopedata	Type definition for scope data structure
xPCAddScope	Create new scope
xPCGetNumScopes	Return number of scopes added to target application
xPCGetNumScSignals	Returns number of signals added to specific scope
xPCGetScope	Get and copy scope data to structure
xPCGetScopeList	Get and copy list of scope numbers
xPCGetScopes	Get and copy list of scope numbers
xPCIsScFinished	Return data acquisition status for scope
xPCRemScope	Remove scope
xPCScAddSignal	Add signal to scope
xPCScGetAutoRestart	Scope autorestart status

xPCScGetData	Copy scope data to array
xPCScGetDecimation	Return decimation of scope
xPCScGetNumPrePostSamples	Get number of pre- or post-triggering samples before triggering scope
xPCScGetNumSamples	Get number of samples in one data acquisition cycle
xPCScGetNumSignals	Get number of signals in scope
xPCScGetSignalList	Copy list of signals to array
xPCScGetSignals	Copy list of signals to array
xPCScGetStartTime	Get start time for last data acquisition cycle
xPCScGetState	Get state of scope
xPCScGetTriggerLevel	Get trigger level for scope
xPCScGetTriggerMode	Get trigger mode for scope
xPCScGetTriggerScope	Get trigger scope
xPCScGetTriggerScopeSample	Get sample number for triggering scope
xPCScGetTriggerSignal	Get trigger signal for scope
xPCScGetTriggerSlope	Get trigger slope for scope
xPCScGetType	Get type of scope
xPCScRemSignal	Remove signal from scope
xPCScSetAutoRestart	Scope autorestart status
xPCScSetDecimation	Set decimation of scope
xPCScSetNumPrePostSamples	Set number of pre- or posttriggering samples before triggering scope
xPCScSetNumSamples	Set number of samples in one data acquisition cycle
xPCScSetTriggerLevel	Set trigger level for scope
xPCScSetTriggerMode	Set trigger mode of scope

xPCScSetTriggerScope	Select scope to trigger another scope
xPCScSetTriggerScopeSample	Set sample number for triggering scope
xPCScSetTriggerSignal	Select signal to trigger scope
xPCScSetTriggerSlope	Set slope of signal that triggers scope
xPCScSoftwareTrigger	Set software trigger of scope
xPCScStart	Start data acquisition for scope
xPCScStop	Stop data acquisition for scope
xPCSetScope	Set properties of scope
xPCTgScGetGrid	Get status of grid line for particular scope
xPCTgScGetMode	Get scope mode for displaying signals
xPCTgScGetViewMode	Get view mode for target computer display
xPCTgScGetYLimits	Copy y-axis limits for scope to array
xPCTgScSetGrid	Set grid mode for scope
xPCTgScSetMode	Set display mode for scope
xPCTgScSetViewMode	Set view mode for scope
xPCTgScSetYLimits	Set y-axis limits for scope

## **Parameters**

xPCGetNumParams	Return number of tunable parameters
xPCGetParam	Get parameter value and copy it to array
xPCGetParamDims	Get row and column dimensions of parameter
xPCGetParamIdx	Return parameter index

xPCGetParamName	Get name of parameter
xPCSetParam	Change value of parameter

## Signals

xPCGetNumSignals	Return number of signals
xPCGetSigIdxfromLabel	Return array of signal indices
xPCGetSigLabelWidth	Return number of elements in signal
xPCGetSignal	Return value of signal
xPCGetSignalIdx	Return index for signal
xPCGetSignalLabel	Copy label of signal to character array
xPCGetSignalName	Copy name of signal to character array
xPCGetSignals	Return vector of signal values
xPCGetSignalWidth	Return width of signal

## Data Logs

lgmode	Type definition for logging options structure
xPCGetLogMode	Return logging mode and increment value for target application
xPCGetNumOutputs	Return number of outputs
xPCGetNumStates	Return number of states
xPCGetOutputLog	Copy output log data to array
xPCGetStateLog	Copy state log values to array
xPCGetTETLog	Copy TET log to array
xPCGetTimeLog	Copy time log to array

xPCMaxLogSamples	Return maximum number of samples that can be in log buffer
xPCNumLogSamples	Return number of samples in log buffer
xPCNumLogWraps	Return number of times log buffer wraps
xPCSetLogMode	Set logging mode and increment value of scope

## File Systems

dirStruct	Type definition for file system folder information structure
diskinfo	Type definition for file system disk information structure
fileinfo	Type definition for file information structure
xPCFSCD	Change current folder on target computer to specified path
xPCFSCloseFile	Close file on target computer
xPCFSDir	Get contents of specified folder on target computer
xPCFSDirItems	Get contents of specified folder on target computer
xPCFSDirSize	Return size of specified folder listing on target computer
xPCFSDirStructSize	Get number of items in folder
xPCFSDiskInfo	Information about target computer file system
xPCFSFileInfo	Return information for open file on target computer
xPCFSGetFileSize	Return size of file on target computer

xPCFSGetPWD	Get current folder of target computer
xPCFSMKDIR	Create new folder on target computer
xPCFSOpenFile	Open file on target computer
xPCFSReadFile	Read open file on target computer
xPCFSRemoveFile	Remove file from target computer
xPCFSRMDIR	Remove folder from target computer
xPCFSScGetFilename	Get name of file for scope
xPCFSScGetWriteMode	Get write mode of file for scope
xPCFSScGetWriteSize	Get block write size of data chunks
xPCFSScSetFilename	Specify name for file to contain signal data
xPCFSScSetWriteMode	Specify when file allocation table entry is updated
xPCFSScSetWriteSize	Specify that memory buffer collect data in multiples of write size
xPCFSWriteFile	Write to file on target computer

## Errors

xPCErrorMsg	Return text description for error message
xPCFSGetError	Get text description for error number on target computer file system
xPCGetLastError	Return constant of last error
xPCSetLastError	Set last error to specific string constant

## C API Error Messages

The header file `matlabroot\toolbox\rtw\targets\xpc\api\xpcapiconst.h` defines these error messages.

<b>Message</b>	<b>Description</b>
ECOMPORTACCFAIL	COM port access failed
ECOMPORTISOPEN	COM port is already opened
ECOMPORTREAD	ReadFile failed while reading from COM port
ECOMPORTWRITE	WriteFile failed while writing to COM port
ECOMTIMEOUT	timeout while receiving: check serial link
EFILEOPEN	Error opening file
EFILEREAD	Error reading file
EFILERENAME	Error renaming file
EFILEWRITE	Error writing file
EINTERNAL	Internal Error
EINVADDR	Invalid IP Address
EINVARGUMENT	Invalid Argument
EINVALIDMODEL	Model name does not match saved value
EINVBAUDRATE	Invalid value for baudrate
EINVCOMMTYP	Invalid communication type
EINVCOMPORT	COM port can only be 0 or 1 (COM1 or COM2)
EINVDECIMATION	Decimation must be positive
EINVFILENAME	Invalid file name
EINVINSTANDALONE	Command not valid for StandAlone
EINVLGDATA	Invalid lgdata structure
EINVLGINCR	Invalid increment for value equidistant logging
EINVLGMODE	Invalid Logging mode
EINVLOGID	Invalid log identifier



<b>Message</b>	<b>Description</b>
EINVNUMPARAMS	Invalid number of parameters
EINVNUMSIGNALS	Invalid number of signals
EINVPARIDX	Invalid parameter index
EINVPORT	Invalid Port Number
EINVSCIDX	Invalid Scope Index
EINVSTYPE	Invalid Scope type
EINVSIGIDX	Invalid Signal index
EINVTRIGMODE	Invalid trigger mode
EINVTRIGSLOPE	Invalid Trigger Slope Value
EINVTRSCIDX	Invalid Trigger Scope index
EINVNUMSAMP	Number of samples must be nonnegative
EINVSTARTVAL	Invalid value for "start"
EINVTFIN	Invalid value for TFinal
EINVTS	Invalid value for Ts (must be between 8e-6 and 10)
EINVWSVER	Invalid Winsock version (1.1 needed)
EINXPCVERSION	Target has an invalid version of xPC Target
ELOADAPPFIRST	Load the application first
ELOGGINGDISABLED	Logging is disabled
EMALFORMED	Malformed message
EMEMALLOC	Memory allocation error
ENODATALOGGED	No data has been logged
ENOERR	No error
ENOFREEPORT	No free Port in C API
ENOMORECHANNELS	No more channels in scope
ENOSPACE	Space not allocated
EOUTPUTLOGDISABLED	Output Logging is disabled

<b>Message</b>	<b>Description</b>
EPARNOTFOUND	Parameter not found
EPARSIZMISMATCH	Parameter Size mismatch
EPINGCONNECT	Could not connect to Ping socket
EPINGPORTOPEN	Error opening Ping port
EPINGSOCKET	Ping socket error
EPORTCLOSED	Port is not open
ERUNSIMFIRST	Run simulation first
ESCFINVALIDFNAME	Invalid filename tag used for dynamic file name
ESCFISNOTAUTO	Autorestart must be enabled for dynamic file names
ESCFNUMISNOTMULT	MaxWriteFileSize must be a multiple of the writesize
ESCTYPENOTTGT	Scope Type is not "Target"
ESIGLABELNOTFOUND	Signal label not found
ESIGLABELNOTUNIQUE	Ambiguous signal label (signal labels are not unique)
ESIGNOTFOUND	Signal not found
ESOCKOPEN	Socket Open Error
ESTARTSIMFIRST	Start simulation first
ESTATELOGDISABLED	State Logging is disabled
ESTOPSCFIRST	Stop scope first
ESTOPSIMFIRST	Stop simulation first
ETCPCONNECT	TCP/IP Connect Error
ETCPREAD	TCP/IP Read Error
ETCPTIMEOUT	TCP/IP timeout while receiving data
ETCPWRITE	TCP/IP Write error
ETETLOGDISABLED	TET Logging is disabled

<b>Message</b>	<b>Description</b>
ETGTMEMALLOC	Target memory allocation failed
ETIMELOGDISABLED	Time Logging is disabled
ETOOMANYSAMPLES	Too Many Samples requested
ETOOMANYSCOPES	Too many scopes are present
ETOOMANYSIGNALS	Too many signals in Scope
EUNLOADAPPFIRST	Unload the application first
EUSEDYNSCOPE	Use DYNAMIC_SCOPE flag at compile time
EWRITEFILE	LoadDLM: WriteFile Error
EWSINIT	WINSOCK: Initialization Error
EWSNOTREADY	Winsock not ready

## **C API Structures and Functions – Alphabetical List**

**Purpose** Type definition for file system folder information structure

**Syntax**

```
typedef struct {
    char      Name[8];
    char      Ext[3];
    char      Day;
    int  Month;
    int  Year;
    int  Hour;
    int  Min;
    int  isDir;
    unsigned long  Size;
} dirStruct;
```

**Fields**

<i>Name</i>	This value contains the name of the file or folder.
<i>Ext</i>	This value contains the file type of the element, if the element is a file ( <i>isDir</i> is 0). If the element is a folder ( <i>isDir</i> is 1), this field is empty.
<i>Day</i>	This value contains the day the file or folder was last modified.
<i>Month</i>	This value contains the month the file or folder was last modified.
<i>Year</i>	This value contains the year the file or folder was last modified.
<i>Hour</i>	This value contains the hour the file or folder was last modified.
<i>Min</i>	This value contains the minute the file or folder was last modified.

# dirStruct

---

<i>isDir</i>	This value indicates if the element is a file (0) or folder (1). If it is a folder, Bytes has a value of 0.
<i>Size</i>	This value contains the size of the file in bytes. If the element is a folder, this value is 0.

**Description** The `dirStruct` structure contains information for a folder in the file system.

**See Also** API function `xPCFSDirItems`

**Purpose**

Type definition for file system disk information structure

**Syntax**

```
typedef struct {
    char        Label[12];
    char        DriveLetter;
    char        Reserved[3];
    unsigned int SerialNumber;
    unsigned int FirstPhysicalSector;
    unsigned int FATType;
    unsigned int FATCount;
    unsigned int MaxDirEntries;
    unsigned int BytesPerSector;
    unsigned int SectorsPerCluster;
    unsigned int TotalClusters;
    unsigned int BadClusters;
    unsigned int FreeClusters;
    unsigned int Files;
    unsigned int FileChains;
    unsigned int FreeChains;
    unsigned int LargestFreeChain;
} diskinfo;
```

**Fields**

<i>Label</i>	This value contains the zero-terminated string that contains the volume label. The string is empty if the volume has no label.
<i>DriveLetter</i>	This value contains the drive letter, in uppercase.
<i>Reserved</i>	Reserved.
<i>SerialNumber</i>	This value contains the volume serial number.
<i>FirstPhysicalSector</i>	This value contains the logical block addressing (LBA) address of the logical drive boot record. For 3.5-inch disks, this value is 0.

<i>FATType</i>	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
<i>FATCount</i>	This value contains the number of FAT partitions on the volume.
<i>MaxDirEntries</i>	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
<i>BytesPerSector</i>	This value contains the sector size. This value is most likely to be 512.
<i>SectorsPerCluster</i>	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
<i>TotalClusters</i>	This value contains the number of file storage clusters on the volume.
<i>BadClusters</i>	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
<i>FreeClusters</i>	This value contains the number of clusters that are currently available for storage.
<i>Files</i>	This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
<i>FileChains</i>	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of <i>Files</i> .



<i>FreeChains</i>	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
<i>LargestFreeChain</i>	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to <code>FreeClusters</code> .

**Description** The `diskinfo` structure contains information for file system disks.

**See Also** API function `xPCFSDiskInfo`

# fileinfo

---

**Purpose** Type definition for file information structure

**Syntax**

```
typedef struct {  
    int         FilePos;  
    int         AllocatedSize;  
    int         ClusterChains;  
    int         VolumeSerialNumber;  
    char        FullName[255];  
}fileinfo;
```

**Fields**

<i>FilePos</i>	This value contains the current file pointer.
<i>AllocatedSize</i>	This value contains the currently allocated file size.
<i>ClusterChains</i>	This value indicates how many separate cluster chains are allocated for the file.
<i>VolumeSerialNumber</i>	This value holds the serial number of the volume the file resides on.
<i>FullName</i>	This value contains a copy of the complete path name of the file. This field is valid only while the file is open.

**Description** The `fileinfo` structure contains information for files in the file system.

**See Also** `xPCFSFileInfo`

---

<b>Purpose</b>	Type definition for logging options structure				
<b>Syntax</b>	<pre>typedef struct {     int    <i>mode</i>;     double <i>incrementvalue</i>; } lgmode;</pre>				
<b>Fields</b>	<table><tr><td><i>mode</i></td><td>This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.</td></tr><tr><td><i>incrementvalue</i></td><td>If you set <i>mode</i> to LGMOD_VALUE for value-equidistant data, this option specifies the increment (difference in amplitude) value between logged data points. A data point is logged only when an output signal or a state changes by <i>incrementvalue</i>.  If you set <i>mode</i> to LGMOD_TIME, <i>incrementvalue</i> is ignored.</td></tr></table>	<i>mode</i>	This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.	<i>incrementvalue</i>	If you set <i>mode</i> to LGMOD_VALUE for value-equidistant data, this option specifies the increment (difference in amplitude) value between logged data points. A data point is logged only when an output signal or a state changes by <i>incrementvalue</i> .  If you set <i>mode</i> to LGMOD_TIME, <i>incrementvalue</i> is ignored.
<i>mode</i>	This value indicates the type of logging you want. Specify LGMOD_TIME for time-equidistant logging. Specify LGMOD_VALUE for value-equidistant logging.				
<i>incrementvalue</i>	If you set <i>mode</i> to LGMOD_VALUE for value-equidistant data, this option specifies the increment (difference in amplitude) value between logged data points. A data point is logged only when an output signal or a state changes by <i>incrementvalue</i> .  If you set <i>mode</i> to LGMOD_TIME, <i>incrementvalue</i> is ignored.				
<b>Description</b>	The lgmode structure specifies data logging options. The <i>mode</i> variable accepts either the numeric values 0 or 1 or their equivalent constants LGMOD_TIME or LGMOD_VALUE from xpcapiconst.h.				
<b>See Also</b>	API functions xPCSetLogMode, xPCGetLogMode				

# scopedata

---

**Purpose** Type definition for scope data structure

**Syntax**

```
typedef struct {
    int    number;
    int    type;
    int    state;
    int    signals[10];
    int    numsamples;
    int    decimation;
    int    triggermode;
    int    numprepostsamples;
    int    triggersignal
    int    triggerscope;
    int    triggerscopesample;
    double triggerlevel;
    int    triggerslope;
} scopedata;
```

**Fields**

<i>number</i>	The scope number.
<i>type</i>	Determines whether the scope is displayed on the host computer or on the target computer. Values are one of the following:  1      Host 2      Target
<i>state</i>	Indicates the scope state. Values are one of the following:  0      Waiting to start 1      Scope is waiting for a trigger 2      Data is being acquired 3      Acquisition is finished 4      Scope is stopped (interrupted)

	5	Scope is preacquiring data
<i>signals</i>		List of signal indices from the target object to display on the scope.
<i>numsamples</i>		Number of contiguous samples captured during the acquisition of a data package.
<i>decimation</i>		A number, N, meaning every Nth sample is acquired in a scope window.
<i>triggermode</i>		Trigger mode for a scope. Values are one of the following:
	0	FreeRun (default)
	1	Software
	2	Signal
	3	Scope
<i>numprepostsamples</i>		If this value is less than 0, this is the number of samples to be saved before a trigger event. If this value is greater than 0, this is the number of samples to skip after the trigger event before data acquisition begins.
<i>triggersignal</i>		If <i>triggermode</i> is 2 (Signal), identifies the block output signal to use for triggering the scope. Identify the signal with a signal index.
<i>triggerscope</i>		If <i>triggermode</i> is 3 (Scope), identifies the scope to use for a trigger. A scope can be set to trigger when another scope is triggered.
<i>triggerscopesample</i>		If <i>triggermode</i> is 3 (Scope), specifies the number of samples to be acquired by the triggering scope before triggering a second scope. This must be a nonnegative value.

<i>triggerlevel</i>	If <i>triggermode</i> is 2 (Signal), indicates the value the signal has to cross to trigger the scope to start acquiring data. The trigger level can be crossed with either a rising or falling signal.
<i>triggerslope</i>	If <i>triggermode</i> is 2 (Signal), indicates whether the trigger is on a rising or falling signal. Values are: 0      Either rising or falling (default) 1      Rising 2      Falling

## Description

The `scopedata` structure holds the data about a scope used in the functions `xPCGetScope` and `xPCSetScope`. In the structure, the fields are as in the various `xPCGetSc*` functions (for example, *state* is as in `xPCScGetState`, *signals* is as in `xPCScGetSignals`, etc.). The signal vector is an array of the signal identifiers, terminated by -1.

## See Also

API functions `xPCSetScope`, `xPCGetScope`, `xPCScGetType`, `xPCScGetState`, `xPCScGetSignals`, `xPCScGetNumSamples`, `xPCScGetDecimation`, `xPCScGetTriggerMode`, `xPCScGetNumPrePostSamples`, `xPCScGetTriggerSignal`, `xPCScGetTriggerScope`, `xPCScGetTriggerLevel`, `xPCScGetTriggerSlope`

**Purpose** Create new scope

**Prototype** `void xPCAddScope(int port, int scType, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scType</i>	Enter the type of scope.
<i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .

**Description** The `xPCAddScope` function creates a new scope on the target computer. For *scType*, scopes can be of type `host` or `target`, depending on the value of *scType*:

- `SCTYPE_HOST` for type `host`
- `SCTYPE_TARGET` for type `target`
- `SCTYPE_FILE` for type `file`

Constants for *scType* are defined in the header file `xpcapiconst.h` as `SCTYPE_HOST`, `SCTYPE_TARGET`, and `SCTYPE_FILE`.

Calling the `xPCAddScope` function with *scNum* having the number of an existing scope produces an error. Use `xPCGetScopes` to find the numbers of existing scopes.

**See Also** API functions `xPCScAddSignal`, `xPCScRemSignal`, `xPCRemScope`, `xPCSetScope`, `xPCGetScope`, `xPCGetScopes`

Target object method `xpctarget.xpc.addscope`

# xPCAverageTET

---

<b>Purpose</b>	Return average task execution time
<b>Prototype</b>	<code>double xPCAverageTET(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCAverageTET</code> function returns the average task execution time (TET) for the target application.
<b>Description</b>	The <code>xPCAverageTET</code> function returns the TET for the target application. You can use this function when the target application is running or when it is stopped.
<b>See Also</b>	API functions <code>xPCMaximumTET</code> , <code>xPCMinimumTET</code> Target object property <code>AvgTET</code>



**Purpose** Close RS-232 or TCP/IP communication connection

**Prototype** `void xPCCloseConnection(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCCloseConnection` function closes the RS-232 or TCP/IP communication channel opened by `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, or `xPCOpenConnection`. Unlike `xPCClosePort`, it preserves the connection information such that a subsequent call to `xPCOpenConnection` succeeds without the need to resupply communication data such as the IP address or port number. To completely close the communication channel, call `xPCDeRegisterTarget`. Calling the `xPCCloseConnection` function followed by calling `xPCDeRegisterTarget` is equivalent to calling `xPCClosePort`.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**See Also** API functions `xPCOpenConnection`, `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCReOpenPort`, `xPCRegisterTarget`, `xPCDeRegisterTarget`

# xPCClosePort

---

**Purpose** Close RS-232 or TCP/IP communication connection

**Prototype** `void xPCClosePort(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCClosePort` function closes the RS-232 or TCP/IP communication channel opened by either `xPCOpenSerialPort` or by `xPCOpenTcpIpPort`. Calling this function is equivalent to calling `xPCCloseConnection` and `xPCDeRegisterTarget`.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**See Also** API functions `xPCOpenSerialPort`, `xPCOpenTcpIpPort`, `xPCReOpenPort`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCRegisterTarget`, `xPCDeRegisterTarget`

Target object method `xpctarget.xpc.close`

**Purpose** Delete target communication properties from xPC Target API library

**Prototype** `void xPCDeRegisterTarget(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCDeRegisterTarget` function causes the xPC Target API library to completely “forget” about the target communication properties. It works similarly to `xPCClosePort`, but does not close the connection to the target machine. Before calling this function, you must first call the function `xPCCloseConnection` to close the connection to the target machine. The combination of calling the `xPCCloseConnection` and `xPCDeRegisterTarget` functions has the same result as calling `xPCClosePort`.

**See Also** API functions `xPCRegisterTarget`, `xPCOpenTcpIpPort`, `xPCOpenSerialPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCTargetPing`

# xPCErrorMsg

---

- Purpose** Return text description for error message
- Prototype** `char *xPCErrorMsg(int error_number, char *error_message);`
- Arguments**
- error\_number* Enter the constant of an error.
  - error\_message* The `xPCErrorMsg` function copies the error message string into the buffer pointed to by *error\_message*. *error\_message* is then returned. You can later use *error\_message* in a function such as `printf`.  
  
If *error\_message* is `NULL`, the `xPCErrorMsg` function returns a pointer to a statically allocated string.
- Return** The `xPCErrorMsg` function returns a string associated with the error *error\_number*.
- Description** The `xPCErrorMsg` function returns *error\_message*, which makes it convenient to use in a `printf` or similar statement. Use the `xPCGetLastError` function to get the constant for which you are getting the message.
- See Also** API functions `xPCSetLastError`, `xPCGetLastError`

**Purpose** Unload xPC Target DLL

**Prototype** `void xPCFreeAPI(void);`

**Description** The xPCFreeAPI function unloads the xPC Target dynamic link library. You must execute this function once at the end of the application to unload the xPC Target API DLL. This frees the memory allocated to the functions. This function is defined in the file `xpcinitfree.c`. Link this file with your application.

**See Also** API functions `xPCInitAPI`, `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

# xPCFSCD

---

**Purpose** Change current folder on target computer to specified path

**Prototype** `void xPCFSCD(int port, char *dir);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dir</i>	Enter the path on the target computer to change to.

**Description** The `xPCFSCD` function changes the current folder on the target computer to the path specified in *dir*. Use the `xPCFSGetPWD` function to show the current folder of the target computer.

**See Also**

- API function `xPCFSGetPWD`
- File object method `xpctarget.fsbase.cd`

**Purpose** Close file on target computer

**Prototype** `void xPCFSCloseFile(int port, int fileHandle);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

**Description** The `xPCFSCloseFile` function closes the file associated with *fileHandle* on the target computer. *fileHandle* is the handle of a file previously opened by the `xPCFSOpenFile` function.

**See Also** API functions `xPCFSOpenFile`, `xPCFSReadFile`, `xPCFSWriteFile`  
File object method `xpctarget.fs.fclose`

# xPCFSDir

---

**Purpose** Get contents of specified folder on target computer

**Prototype** `void xPCFSDir(int port, const char *path, char *data, int numbytes);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	Enter the path on the target computer.
<i>data</i>	The contents of the folder are stored in <i>data</i> , whose allocated size is specified in <i>numbytes</i> .
<i>numbytes</i>	Enter the size, in bytes, of the array <i>data</i> .

**Description** The `xPCFSDir` function copies the contents of the target computer folder specified by *path* into *data*. The `xPCFSDir` function returns the listing in the *data* array, which must be of size *numbytes*. Use the `xPCFSDirSize` function to obtain the size of the folder listing for the *numbytes* parameter.

**See Also** API function `xPCFSDirSize`  
File object method `xpctarget.fsbase.dir`



**Purpose**

Get contents of specified folder on target computer

**Prototype**

```
void xPCFSDirItems(int port, const char *path, dirStruct  
*dirs, int numDirItems);
```

**Arguments**

*port* Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

*path* Enter the path on the target computer.

*dirs* Enter the structure to contain the contents of the folder.

*numDirItems* Enter the number of items in the folder.

**Description**

The xPCFSDirItems function copies the contents of the target computer folder specified by *path*. The xPCFSDirItems function copies the listing into the *dirs* structure, which must be of size *numDirItems*. Use the xPCFSDirStructSize function to obtain the size of the folder for the *numDirItems* parameter.

**See Also**

API functions xPCFSDirStructSize, dirStruct  
File object method xpctarget.fsbasedir

# xPCFSDirSize

---

**Purpose** Return size of specified folder listing on target computer

**Prototype** `int xPCFSDirSize(int port, const char *path);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	Enter the folder path on the target computer.

**Return** The `xPCFSDirSize` function returns the size, in bytes, of the specified folder listing. If this function detects an error, it returns -1.

**Description** The `xPCFSDirSize` function returns the size, in bytes, of the buffer required to list the folder contents on the target computer. Use this size as the *numbytes* parameter in the `xPCFSDir` function.

**See Also** API function `xPCFSDirItems`  
File object method `xpctarget.fsbase.dir`

**Purpose** Get number of items in folder

**Prototype** `int xPCFSDirStructSize(int port, const char *path);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>path</i>	Enter the folder path on the target computer.

**Return** The `xPCFSDirStructSize` function returns the number of items in the folder on the target computer. If this function detects an error, it returns -1.

**Description** The `xPCFSDirStructSize` function returns the number of items in the folder on the target computer. Use this size as the *numDirItems* parameter in the `xPCFSDirItems` function.

**See Also** API function `xPCFSDir`  
File object method `xpctarget.fsbase.dir`

# xPCFSDiskInfo

---

**Purpose** Information about target computer file system

**Prototype** `diskinfo xPCFSDiskInfo(int port, const char *driveletter);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>driveletter</i>	Enter the drive letter of the file system for which you want information.

**Description** The `xPCFSDiskInfo` function returns disk information for the file system of the specified target computer drive, *driveletter*. This function returns this information in the `diskinfo` structure.

**See Also** API structure `xpctarget.fs.diskinfo`

<b>Purpose</b>	Return information for open file on target computer				
<b>Prototype</b>	<code>fileinfo xPCFSFileInfo(int <i>port</i>, int <i>fileHandle</i>);</code>				
<b>Arguments</b>	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>fileHandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>fileHandle</i>	Enter the file handle of an open file on the target computer.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.				
<b>Description</b>	The <code>xPCFSFileInfo</code> function returns information about the specified open file, <code>filehandle</code> , in a structure of type <code>fileinfo</code> .				
<b>See Also</b>	Structure <code>xpctarget.fs.fileinfo</code>				

# xPCFSError

---

**Purpose** Get text description for error number on target computer file system

**Prototype** `void xPCFSError(int port, unsigned int error_number, char *error_message);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>error_number</i>	Enter the constant of an error.
<i>error_message</i>	The string of the message associated with the error <i>error_number</i> is stored in <i>error_message</i> .

**Description** The `xPCFSError` function gets the *error\_message* associated with *error\_number*. This enables you to use the error message in a `printf` or similar statement.

**Purpose** Return size of file on target computer

**Prototype** `int xPCFSGetFileSize(int port, int fileHandle);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.

**Return** Return the size of the specified file in bytes. If this function detects an error, it returns -1.

**Description** The `xPCFSGetFileSize` function returns the size, in bytes, of the file associated with *fileHandle* on the target computer. *fileHandle* is the handle of a file previously opened by the `xPCFSOpenFile` function.

**See Also** API functions `xPCFSOpenFile`, `xPCFSReadFile`  
File object methods `xpctarget.fs.fopen`, `xpctarget.fs.fread`

# xPCFSGetPWD

---

**Purpose** Get current folder of target computer

**Prototype** `void xPCFSGetPWD(int port, char *pwd);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>pwd</i>	The path of the current folder is stored in <i>pwd</i> .

**Description** The `xPCFSGetPWD` function places the path of the current folder on the target computer in *pwd*, which must be allocated by the caller.

**See Also** File object method `xpctarget.fsbase.pwd`



**Purpose** Create new folder on target computer

**Prototype** `void xPCFSMKDIR(int port, const char *dirname);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dirname</i>	Enter the name of the folder to create on the target computer.

**Description** The `xPCFSMKDIR` function creates the folder *dirname* in the current folder of the target computer.

**See Also**

- API function `xPCFSGetPWD`
- File object method `xpctarget.fsbase.mkdir`

# xPCFSOpenFile

---

**Purpose** Open file on target computer

**Prototype** `int xPCFSOpenFile(int port, const char *filename, const char *permission);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>filename</i>	Enter the name of the file to open on the target computer.
<i>permission</i>	Enter the read/write permission with which to open the file. Values are <code>r</code> (read) or <code>w</code> (read/write).

**Return** The `xPCFSOpenFile` function returns the file handle for the opened file. If function detects an error, it returns `-1`.

**Description** The `xPCFSOpenFile` function opens the specified file, *filename*, on the target computer. If the file does not exist, the `xPCFSOpenFile` function creates *filename*, then opens it. You can open a file for read or read/write access.

**See Also** API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSReadFile`, `xPCFSWriteFile`  
File object methods `xpctarget.fs.fclose`, `xpctarget.fs.filetable`, `xpctarget.fs.fopen`, `xpctarget.fs.fread`, `xpctarget.fs.fwrite`

**Purpose** Read open file on target computer

**Prototype** `void xPCFSReadFile(int port, int fileHandle, int start, int numbytes, unsigned char *data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.
<i>start</i>	Enter an offset from the beginning of the file from which this function can start to read.
<i>numbytes</i>	Enter the number of bytes this function is to read from the file.
<i>data</i>	The contents of the file are stored in <i>data</i> .

**Description** The `xPCFSReadFile` function reads an open file on the target computer and places the results of the read operation in the array *data*. *fileHandle* is the file handle of a file previously opened by `xPCFSOpenFile`. You can specify that the read operation begin at the beginning of the file (default) or at a certain offset into the file (*start*). The *numbytes* parameter specifies how many bytes the `xPCFSReadFile` function is to read from the file.

**See Also** API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSWriteFile`  
File object methods `xpctarget.fs.fopen`, `xpctarget.fs.fread`

# xPCFSRemoveFile

---

**Purpose** Remove file from target computer

**Prototype** `void xPCFSRemoveFile(int port, const char *filename);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>filename</i>	Enter the name of a file on the target computer.

**Description** The `xPCFSRemoveFile` function removes the file named *filename* from the target computer file system. *filename* can be a relative or absolute path name on the target computer.

**See Also** File object method `xpctarget.fs.removefile`

**Purpose** Remove folder from target computer

**Prototype** `void xPCFSRMDIR(int port, const char *dirname);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>dirname</i>	Enter the name of a folder on the target computer.

**Description** The `xPCFSRMDIR` function removes a folder named *dirname* from the target computer file system. *dirname* can be a relative or absolute path-name on the target computer.

**See Also** File object method `xpctarget.fsbase.rmdir`

# xPCFSScGetFilename

---

**Purpose** Get name of file for scope

**Prototype** `const char *xPCFSScGetFilename(int port, int scNum, char *filename);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>filename</i>	The name of the file for the specified scope is stored in <i>filename</i> .

**Return** Returns the value of *filename*, the name of the file for the scope.

**Description** The `xPCFSScGetFilename` function returns the name of the file to which scope *scNum* will save signal data. *filename* points to a caller-allocated character array to which the filename is copied.

**See Also** API function `xPCFSScSetFilename`  
Scope object property `Filename`

**Purpose** Get write mode of file for scope

**Prototype** `int xPCFSScGetWriteMode(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** Returns the number indicating the write mode. Values are

- |   |  |
|---|--|
| 0 | Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact). |
| 1 | Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.   |

**Description** The `xPCFSScGetWriteMode` function returns the write mode of the file for the scope.

**See Also** API function `xPCFSScSetWriteMode`  
Scope object property `Mode`

# xPCFSScGetWriteSize

---

**Purpose** Get block write size of data chunks

**Prototype** `unsigned int xPCFSScGetWriteSize(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** Returns the block size, in bytes, of the data chunks.

**Description** The `xPCFSScGetWriteSize` function gets the block size, in bytes, of the data chunks.

**See Also** API function `xPCFSScSetWriteSize`  
Scope object property `WriteSize`



**Purpose** Specify name for file to contain signal data

**Prototype** `void xPCFSScSetFilename(int port, int scNum,  
const char *filename);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>filename</i>	Enter the name of a file to contain the signal data.

**Description** The `xPCFSScSetFilename` function sets the name of the file to which the scope will save the signal data. The xPC Target software creates this file in the target computer file system. Note that you can only call this function when the scope is stopped.

**See Also** API function `xPCFSScGetFilename`  
Scope object property `Filename`

# xPCFSScSetWriteMode

---

**Purpose** Specify when file allocation table entry is updated

**Prototype** `void xPCFSScSetWriteMode(int port, int scNum, int writeMode);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeMode</i>	Enter an integer for the write mode: 0 Enables lazy write mode 1 Enables commit write mode

**Description** The `xPCFSScSetWriteMode` function specifies when a file allocation table (FAT) entry is updated. Both modes write the signal data to the file, as follows:

- 0 Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
- 1 Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.

**See Also** API function `xPCFSScGetWriteMode`  
Scope object property `Mode`

**Purpose** Specify that memory buffer collect data in multiples of write size

**Prototype** `void xPCFSScSetWriteSize(int port, int scNum, unsigned int writeSize);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>writeSize</i>	Enter the block size, in bytes, of the data chunks.

**Description** The `xPCFSScSetWriteSize` function specifies that a memory buffer collect data in multiples of *writeSize*. By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance. *writeSize* must be a multiple of 512.

**See Also** API function `xPCFSScGetWriteSize`  
Scope object property `WriteSize`

# xPCFSWriteFile

---

**Purpose** Write to file on target computer

**Prototype** `void xPCFSWriteFile(int port, int fileHandle, int numbytes, const unsigned char *data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>fileHandle</i>	Enter the file handle of an open file on the target computer.
<i>numbytes</i>	Enter the number of bytes this function is to write into the file.
<i>data</i>	The contents to write to <i>fileHandle</i> are stored in <i>data</i> .

**Description** The `xPCFSWriteFile` function writes the contents of the array *data* to the file specified by *fileHandle* on the target computer. The *fileHandle* parameter is the handle of a file previously opened by `xPCFSOpenFile`. *numbytes* is the number of bytes to write to the file.

**See Also** API functions `xPCFSCloseFile`, `xPCFSGetFileSize`, `xPCFSOpenFile`, `xPCFSReadFile`

<b>Purpose</b>	Get version number of xPC Target API
<b>Prototype</b>	<code>const char *xPCGetAPIVersion(void);</code>
<b>Return</b>	The <code>xPCGetApiVersion</code> function returns a string with the version number of the xPC Target kernel on the target computer.
<b>Description</b>	The <code>xPCGetApiVersion</code> function returns a string with the version number of the xPC Target kernel on the target computer. The string is a constant string within the API DLL. Do not modify this string.
<b>See Also</b>	API function <code>xPCGetTargetVersion</code>

# xPCGetAppName

---

**Purpose** Return target application name

**Prototype** `char *xPCGetAppName(int port, char *model_name);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>model_name</i>	The <code>xPCGetAppName</code> function copies the target application name string into the buffer pointed to by <i>model_name</i> . <i>model_name</i> is then returned. You can later use <i>model_name</i> in a function such as <code>printf</code> .  Note that the maximum size of the buffer is 256 bytes. To reserve enough space for the application name string, allocate a buffer of size 256 bytes.

**Return** The `xPCGetAppName` function returns a string with the name of the target application.

**Description** The `xPCGetAppName` function returns the name of the target application. You can use the return value, *model\_name*, in a `printf` or similar statement. In case of error, the name string is unchanged.

**Examples** Allocate 256 bytes for the buffer `appname`.

```
char *appname=malloc(256);
xPCGetAppName(iport, appname);
appname=realloc(appname, strlen(appname)+1);
...
free(appname);
```

**See Also** API function `xPCIsAppRunning`  
Target object property `Application`

**Purpose** Return display mode for target message window

**Prototype** `int xPCGetEcho(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Return** The `xPCGetEcho` function returns the number indicating the display mode. Values are

1 Display is on. Messages are displayed in the message display window on the target.

0 Display is off.

**Return** The `xPCGetEcho` function the display mode of the target computer using communication channel *port*. If the function detects an error, it returns -1.

**Description** The `xPCGetEcho` function returns the display mode of the target computer using communication channel *port*. Messages include the status of downloading the target application, changes to parameters, and changes to scope signals.

**See Also** API function `xPCSetEcho`

# xPCGetExecTime

---

**Purpose** Return target application execution time

**Prototype** `double xPCGetExecTime(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Return** The `xPCGetExecTime` function returns the current execution time for a target application. If the function detects an error, it returns -1.

**Description** The `xPCGetExecTime` function returns the current execution time for the running target application. If the target application is stopped, the value is the last running time when the target application was stopped. If the target application is running, the value is the current running time.

**See Also** API functions `xPCSetStopTime`, `xPCGetStopTime`  
Target object property `ExecTime`



<b>Purpose</b>	Return constant of last error
<b>Prototype</b>	<code>int xPCGetLastError(void);</code>
<b>Return</b>	The xPCGetLastError function returns the error constant for the last reported error. If the function did not detect an error, it returns 0.
<b>Description</b>	The xPCGetLastError function returns the constant of the last reported error by another API function. This value is reset every time you call a new function. Therefore, you should check this constant value immediately after a call to an API function. For a list of error constants and messages, see “C API Error Messages” on page 2-10.
<b>See Also</b>	API functions xPCErrorMsg, xPCSetLastError

# xPCGetLoadTimeOut

---

**Purpose** Return timeout value for communication between host computer and target computer

**Prototype** `int xPCGetLoadTimeOut(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Return** The `xPCGetLoadTimeOut` function returns the number of seconds allowed for the communication between the host computer and target application. If the function detects an error, it returns -1.

**Description** The `xPCGetLoadTimeOut` function returns the number of seconds allowed for the communication between the host computer and the target application. When an xPC Target API function initiates communication between the host computer and target computer, it waits for a certain amount of time before checking to see if the communication is complete. In the case where communication with the target computer is not complete, the function returns a timeout error.

For example, when you load a new target application onto the target computer, the function `xPCLoadApp` waits for a certain amount of time before checking to see if the initialization of the target application is complete. In the case where initialization of the target application is not complete, the function `xPCLoadApp` returns a timeout error. By default, `xPCLoadApp` checks for the readiness of the target computer for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout is generated. Other functions that communicate with the target computer will wait for *timeOut* seconds before declaring a timeout event. The function `xPCSetLoadTimeOut` sets the timeout to a different number.

Use the `xPCGetLoadTimeOut` function if you suspect that the current number of seconds (the timeout value) is too short. Then use the `xPCSetLoadTimeOut` function to set the timeout to a higher number.

### **See Also**

API functions xPCLoadApp, xPCSetLoadTimeOut

xPCUnloadApp

“Increase the Time for Downloads”

# xPCGetLogMode

---

<b>Purpose</b>	Return logging mode and increment value for target application
<b>Prototype</b>	<code>lgmode xPCGetLogMode(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetLogMode</code> function returns the logging mode in the <code>lgmode</code> structure. If the logging mode is 1 ( <code>LGMOD_VALUE</code> ), this function also returns an increment value in the <code>lgmode</code> structure. If an error occurs, this function returns -1.
<b>Description</b>	The <code>xPCGetLogMode</code> function gets the logging mode and increment value for the current target application. The increment (difference in amplitude) value is measured between logged data points. A data point is logged only when an output signal or a state changes by the increment value.
<b>See Also</b>	API function <code>xPCSetLogMode</code> API structure <code>lgmode</code>

<b>Purpose</b>	Return number of outputs
<b>Prototype</b>	<code>int xPCGetNumOutputs(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetNumOutputs</code> function returns the number of outputs in the current target application. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCGetNumOutputs</code> function returns the number of outputs in the target application. The number of outputs equals the sum of the input signal widths of the output blocks at the root level of the Simulink model.
<b>See Also</b>	API functions <code>xPCGetOutputLog</code> , <code>xPCGetNumStates</code> , <code>xPCGetStateLog</code>

# xPCGetNumParams

---

<b>Purpose</b>	Return number of tunable parameters
<b>Prototype</b>	<code>int xPCGetNumParams(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetNumParams</code> function returns the number of tunable parameters in the target application. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCGetNumParams</code> function returns the number of tunable parameters in the target application. Use this function to see how many parameters you can get or modify.
<b>See Also</b>	API functions <code>xPCGetParamIdx</code> , <code>xPCSetParam</code> , <code>xPCGetParam</code> , <code>xPCGetParamName</code> , <code>xPCGetParamDims</code> Target object property <code>NumParameters</code>

<b>Purpose</b>	Return number of scopes added to target application
<b>Prototype</b>	<code>int xPCGetNumScopes(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetNumScopes</code> function returns the number of scopes that have been added to the target application. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCGetNumScopes</code> function returns the number of scopes that have been added to the target application.

# xPCGetNumScSignals

---

**Purpose** Returns number of signals added to specific scope

**Prototype** `int xPCGetNumScSignals(int port, int scopeId);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scopeId</i>	Enter the ID number of the scope for which you want to get the number of added signals.

**Return** The `xPCGetNumScSignals` function returns the number of signals that have been added to the scope, *scopeID*. If the function detects an error, it returns -1.

**Description** The `xPCGetNumScSignals` function returns the number of signals that have been added to the scope, *scopeID*.



<b>Purpose</b>	Return number of signals
<b>Prototype</b>	<code>int xPCGetNumSignals(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetNumSignals</code> function returns the number of signals in the target application. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCGetNumSignals</code> function returns the total number of signals in the target application that can be monitored from the host. Use this function to see how many signals you can monitor.
<b>See Also</b>	API functions <code>xPCGetSignalIdx</code> , <code>xPCGetSignal</code> , <code>xPCGetSignals</code> , <code>xPCGetSignalName</code> , <code>xPCGetSignalWidth</code> Target object property <code>NumSignals</code>

# xPCGetNumStates

---

<b>Purpose</b>	Return number of states
<b>Prototype</b>	<code>int xPCGetNumStates(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetNumStates</code> function returns the number of states in the target application. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCGetNumStates</code> function returns the number of states in the target application.
<b>See Also</b>	API functions <code>xPCGetStateLog</code> , <code>xPCGetNumOutputs</code> , <code>xPCGetOutputLog</code> Target object property <code>StateLog</code>

**Purpose** Copy output log data to array

**Prototype**

```
void xPCGetOutputLog(int port, int first_sample,  
int num_samples,  
int decimation, int output_id, double *output_data);
```

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the output log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>output_id</i>	Enter an output identification number.
<i>output_data</i>	The log is stored in <i>output_data</i> , whose allocation is the responsibility of the caller.

**Description** The `xPCGetOutputLog` function gets the output log and copies that log to an array. You get the data for each output signal in turn by specifying *output\_id*. Output IDs range from 0 to (N-1), where N is the return value of `xPCGetNumOutputs`. Entering 1 for *decimation* copies all values. Entering N copies every Nth value.

For *first\_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Get the maximum number of samples by calling the function `xPCNumLogSamples`.

Note that the target application must be stopped before you get the number.

# xPCGetOutputLog

---

## See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumOutputs`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Target object method `xpctarget.xpc.getlog`

Target object property `OutputLog`

**Purpose** Get parameter value and copy it to array

**Prototype** `void xPCGetParam(int port, int paramIndex, double *paramValue);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIndex</i>	Enter the index for a parameter.
<i>paramValue</i>	The function returns a parameter value as an array of doubles.

**Description** The `xPCGetParam` function returns the parameter as an array in *paramValue*. *paramValue* must be large enough to hold the parameter. You can query the size by calling the function `xPCGetParamDims`. Get the parameter index by calling the function `xPCGetParamIdx`. The parameter matrix is returned as a vector, with the conversion being done in column-major format. It is also returned as a double, regardless of the data type of the actual parameter.

For *paramIndex*, values range from 0 to (N-1), where N is the return value of `xPCGetNumParams`.

**See Also** API functions `xPCSetParam`, `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetNumParams`

Target object method `xpctarget.xpc.getparamid`

Target object properties `ShowParameters`, `Parameters`

# xPCGetParamDims

---

**Purpose** Get row and column dimensions of parameter

**Prototype** `void xPCGetParamDims(int port, int paramIndex, int *dimension);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIndex</i>	Parameter index.
<i>dimension</i>	Dimensions (row, column) of a parameter.

**Description** The `xPCGetParamDims` function gets the dimensions (row, column) of a parameter with *paramIndex* and stores them in *dimension*, which must have at least two elements.

For *paramIndex*, values range from 0 to (N-1), where N is the return value of `xPCGetNumParams`.

**See Also** API functions `xPCGetParamIdx`, `xPCGetParamName`, `xPCSetParam`, `xPCGetParam`, `xPCGetNumParams`

Target object method `xpctarget.xpc.getparamid`

Target object properties `ShowParameters`, `Parameters`

**Purpose** Return parameter index

**Prototype** `int xPCGetParamIdx(int port, const char *blockName, const char *paramName);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>blockName</i>	Enter the full block path generated by Simulink Coder.
<i>paramName</i>	Enter the parameter name for a parameter associated with the block.

**Return** The `xPCGetParamIdx` function returns the parameter index for the parameter name. If the function detects an error, it returns -1.

**Description** The `xPCGetParamIdx` function returns the parameter index for the parameter name (*paramName*) associated with a Simulink block (*blockName*). Both *blockName* and *paramName* must be identical to those generated at target application building time. The block names should be referenced from the file `model_namept.m` in the generated code, where *model\_name* is the name of the model. Note that a block can have one or more parameters.

**See Also** API functions `xPCGetParamDims`, `xPCGetParamName`, `xPCGetParam`  
Target object method `xpctarget.xpc.getparamid`  
Target object properties `ShowParameters`, `Parameters`

# xPCGetParamName

---

**Purpose** Get name of parameter

**Prototype**

```
void xPCGetParamName(int port, int paramIdx,  
char *blockName, char  
*paramName);
```

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIdx</i>	Enter a parameter index.
<i>blockName</i>	String with the full block path generated by Simulink Coder.
<i>paramName</i>	Name of a parameter for a specific block.

**Description** The `xPCGetParamName` function gets the parameter name and block name for a parameter with the index *paramIdx*. The block path and name are returned and stored in *blockName*, and the parameter name is returned and stored in *paramName*. You must allocate enough space for both *blockName* and *paramName*. If the *paramIdx* is invalid, `xPCGetLastError` returns nonzero, and the strings are unchanged. Get the parameter index from the function `xPCGetParamIdx`.

**See Also** API functions `xPCGetParam`, `xPCGetParamDims`, `xPCGetParamIdx`  
Target object properties `ShowParameters`, `Parameters`



<b>Purpose</b>	Return target application sample time
<b>Prototype</b>	<code>double xPCGetSampleTime(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCGetSampleTime</code> function returns the sample time, in seconds, of the target application. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCGetSampleTime</code> function returns the sample time, in seconds, of the target application. You can get the error by using the function <code>xPCGetLastError</code> .
<b>See Also</b>	API function <code>xPCSetSampleTime</code> Target object property <code>SampleTime</code>

# xPCGetScope

---

**Purpose** Get and copy scope data to structure

**Prototype** `scopedata xPCGetScope(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCGetScope` function returns a structure of type `scopedata`.

**Description** The `xPCGetScope` function gets properties of a scope with *scNum* and copies the properties into a structure with type `scopedata`. You can use this function in conjunction with `xPCSetScope` to change several properties of a scope at one time. See `scopedata` for a list of properties. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCSetScope`, `scopedata`  
Target object method `xpctarget.xpc.getscope`

**Purpose** Get and copy list of scope numbers

**Prototype** `void xPCGetScopeList(int port, int *data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers.

**Description** The `xPCGetScopeList` function gets the list of scopes currently defined. *data* must be large enough to hold the list of scopes. You can query the size by calling the function `xPCGetNumScopes`.

---

**Note** Use the `xPCGetScopeList` function instead of the `xPCGetScopes` function. The `xPCGetScopes` will be obsoleted in a future release.

---

# xPCGetScopes

---

**Purpose** Get and copy list of scope numbers

**Prototype** `void xPCGetScopes(int port, int *data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	List of scope numbers in an integer array (allocated by the caller) as a list of unsorted integers and terminated by -1.

**Description** The `xPCGetScopes` function gets the list of scopes currently defined. You can use the constant `MAX_SCOPES` (defined in `xpcapiconst.h`) as the size of *data*. This is currently set to 30 scopes.

---

**Note** This function will be obsoleted in a future release. Use the `xPCGetScopeList` function instead.

---

**See Also** API functions `xPCSetScope`, `xPCGetScope`, `xPCScGetSignals`  
Target object property `Scopes`

<b>Purpose</b>	Return length of time xPC Target kernel has been running		
<b>Prototype</b>	<code>double xPCGetSessionTime(int <i>port</i>);</code>		
<b>Arguments</b>	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .		
<b>Return</b>	The <code>xPCGetSessionTime</code> function returns the amount of time in seconds that the xPC Target kernel has been running on the target computer. If the function detects an error, it returns -1.		
<b>Description</b>	The <code>xPCGetSessionTime</code> function returns, as a double, the amount of time in seconds that the xPC Target kernel has been running. This value is also the time that has elapsed since you last booted the target computer.		

# xPCGetSignal

---

**Purpose** Return value of signal

**Prototype** `double xPCGetSignal(int port, int sigNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigNum</i>	Enter a signal number.

**Return** The `xPCGetSignal` function returns the current value of signal *sigNum*. If the function detects an error, it returns -1.

**Description** The `xPCGetSignal` function returns the current value of a signal. For vector signals, use `xPCGetSignals` rather than call this function multiple times. Use the `xPCGetSignalIdx` function to get the signal number.

**See Also** API function `xPCGetSignals`  
Target object properties `ShowSignals`, `Signals`

<b>Purpose</b>	Return index for signal				
<b>Prototype</b>	<code>int xPCGetSignalIdx(int <i>port</i>, const char *<i>sigName</i>);</code>				
<b>Arguments</b>	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>sigName</i></td><td>Enter a signal name.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>sigName</i>	Enter a signal name.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>sigName</i>	Enter a signal name.				
<b>Return</b>	The <code>xPCGetSignalIdx</code> function returns the index for the signal with name <i>sigName</i> . If the function detects an error, it returns -1.				
<b>Description</b>	The <code>xPCGetSignalIdx</code> function returns the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file <code>model_namebio.m</code> in the generated code, where <i>model_name</i> is the name of the model. The creator of the application should already know the signal name.				
<b>See Also</b>	API functions <code>xPCGetSignalName</code> , <code>xPCGetSignalWidth</code> , <code>xPCGetSignal</code> , <code>xPCGetSignals</code> Target object method <code>xpctarget.xpc.getsignalid</code>				

# xPCGetSigIdxfromLabel

---

**Purpose** Return array of signal indices

**Prototype** `int xPCGetSigIdxfromLabel(int port, const char *sigLabel, int *sigIds);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigLabel</i>	String with the name of a signal label.
<i>sigIds</i>	Return array of signal indices.

**Return** If `xPCGetSigIdxfromLabel` finds a signal, it fills an array *sigIds* with signal indices and returns 0. If it finds no signal, it returns -1.

**Description** The `xPCGetSigIdxfromLabel` function returns in *sigIds* the array of signal indices for signal *sigName*. This function assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

*sigIds* must be large enough to contain the array of indices. You can use the `xPCGetSigLabelWidth` function to get the required amount of memory to be allocated by the *sigIds* array.

**See Also** API functions `xPCGetSignalLabel`, `xPCGetSigLabelWidth`



**Purpose**

Copy label of signal to character array

**Prototype**

```
char * xPCGetSignalLabel(int port, int sigIdx,  
char *sigLabel);
```

**Arguments**

*port*            Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*sigIdx*          Enter signal index.

*sigLabel*        Return signal label associated with signal index, *sigIdx*.

**Return**

The `xPCGetSignalLabel` function returns the label of the signal.

**Description**

The `xPCGetSignalLabel` function copies and returns the signal label, including the block path, of a signal with *sigIdx*. The result is stored in *sigLabel*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigLabel* is unchanged. The function returns *sigLabel*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index. Signal labels must be unique.

This function assumes that you have labeled the signal for which you request the index (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.

**See Also**

API functions `xPCGetSigIdxfromLabel`, `xPCGetSigLabelWidth`

# xPCGetSigLabelWidth

---

**Purpose** Return number of elements in signal

**Prototype** `int xPCGetSigLabelWidth(int port, const char *sigName);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigName</i>	String with the name of a signal.

**Return** The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. If the function detects an error, it returns -1.

**Description** The `xPCGetSigLabelWidth` function returns the number of elements that the signal `sigName` contains. This function assumes that you have labeled the signal for which you request the elements (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label. Signal labels must be unique.

**See Also** API functions `xPCGetSigIdxfromLabel`, `xPCGetSignalLabel`

**Purpose** Copy name of signal to character array

**Prototype** `char *xPCGetSignalName(int port, int sigIdx,  
char *sigName);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter a signal index.
<i>sigName</i>	String with the name of a signal.

**Return** The `xPCGetSignalName` function returns the name of the signal.

**Description** The `xPCGetSignalName` function copies and returns the signal name, including the block path, of a signal with *sigIdx*. The result is stored in *sigName*. If *sigIdx* is invalid, `xPCGetLastError` returns a nonzero value, and *sigName* is unchanged. The function returns *sigName*, which makes it convenient to use in a `printf` or similar statement. This function assumes that you already know the signal index.

**See Also** API functions `xPCGetSignalIdx`, `xPCGetSignalWidth`, `xPCGetSignal`, `xPCGetSignals`

Target object properties `ShowSignals`, `Signals`

# xPCGetSignals

---

**Purpose** Return vector of signal values

**Prototype**

```
int xPCGetSignals(int port, int numSignals,
const int *signals,
double *values);
```

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>numSignals</i>	Enter the number of signals to be acquired (that is, the number of values in <i>signals</i> ).
<i>signals</i>	Enter the list of signal numbers to be acquired.
<i>values</i>	Returned values are stored in the double array <i>values</i> .

**Return** The `xPCGetSignals` function returns 0 if it completes execution without detecting an error. If the function detects an error, it returns -1.

**Description** The `xPCGetSignals` function is the vector version of the function `xPCGetSignal`. This function returns the values of a vector of signals (up to 1000) as fast as it can acquire them. The signal values may not be at the same time step (for that, define a scope of type `SCTYPE_HOST` and use `xPCScGetData`). `xPCGetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same result. However, the `xPCGetSignals` function is faster, and the signal values are more likely to be spaced closely together. The signals are converted to doubles regardless of the actual data type of the signal.

For *signals*, the list you provide should be stored in an integer array. Get the signal numbers with the function `xPCGetSignalIdx`.

**See Also** API function `xPCGetSignal`, `xPCGetSignalIdx`

**Example** To reference signal vector data rather than scalar values, pass a vector of indices for the signal data. For example:

```

/*****/

/* Assume a signal of width 10, with the blockpath
 * mySubsys/mySignal and the signal index s1.
 */

int i;
int sigId[10];
double sigVal[10]; /* Signal values are stored here */

/* Get the ID of the first signal */
sigId[0] = xPCGetSignalIdx(port, "mySubsys/mySignal/s1");

if (sigId[0] == -1) {
/* Handle error */
}

for (i = 1; i < 10; i++) {
    sigId[i] = sigId[0] + i;
}

xPCGetSignals(port, 10, sigId, sigVal);
/* If no error, sigVal should have the signal values */

/*****/
```

To repeatedly get the signals, repeat the call to `xPCGetSignals`. If you do not change `sigID`, you only need to call `xPCGetSignalIdx` once.

# xPCGetSignalWidth

---

**Purpose** Return width of signal

**Prototype** `int xPCGetSignalWidth(int port, int sigIdx);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>sigIdx</i>	Enter the index of a signal.

**Return** The `xPCGetSignalWidth` function returns the signal width for a signal with *sigIdx*. If the function detects an error, it returns -1.

**Description** The `xPCGetSignalWidth` function returns the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector again. A signal's width is the number of signals in the vector.

**See Also** API functions `xPCGetSignalIdx`, `xPCGetSignalName`, `xPCGetSignal`, `xPCGetSignals`

**Purpose** Copy state log values to array

**Prototype**

```
void xPCGetStateLog(int port, int first_sample,  
int num_samples,  
int decimation, int state_id, double *state_data);
```

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the output log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>state_id</i>	Enter a state identification number.
<i>state_data</i>	The log is stored in <i>state_data</i> , whose allocation is the responsibility of the caller.

**Description** The `xPCGetStateLog` function gets the state log. It then copies the log into *state\_data*. You get the data for each state signal in turn by specifying the *state\_id*. State IDs range from 1 to (N-1), where N is the return value of `xPCGetNumStates`. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first\_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

# xPCGetStateLog

---

## See Also

API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumStates`, `xPCGetOutputLog`, `xPCGetTETLog`, `xPCGetTimeLog`

Target object method `xpctarget.xpc.getlog`

Target object property `StateLog`



**Purpose** Return stop time

**Prototype** `double xPCGetStopTime(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Return** The `xPCGetStopTime` function returns the stop time as a double, in seconds, of the target application. If the function detects an error, it returns `-10.0`. If the stop time is infinity (run forever), this function returns `-1.0`.

**Description** The `xPCGetStopTime` function returns the stop time, in seconds, of the target application. This is the amount of time the target application runs before stopping. If the function detects an error, it returns `-10.0`. You will then need to use the function `xPCGetLastError` to find the error number.

**See Also** API function `xPCSetStopTime`  
Target object property `StopTime`

# xPCGetTargetVersion

---

**Purpose** Get xPC Target kernel version

**Prototype** `void xPCGetTargetVersion(int port, char *ver);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>ver</i>	The version is stored in <i>ver</i> .

**Description** The `xPCGetTargetVersion` function gets a string with the version number of the xPC Target kernel on the target computer. It then copies that version number into *ver*.

**See Also** `xPCGetAPIVersion`

**Purpose** Copy TET log to array

**Prototype**

```
void xPCGetTETLog(int port, int first_sample,
int num_samples, int decimation,
double *TET_data);
```

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the TET log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>TET_data</i>	The log is stored in <i>TET_data</i> , whose allocation is the responsibility of the caller.

**Description** The `xPCGetTETLog` function gets the task execution time (TET) log. It then copies the log into *TET\_data*. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first\_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

**See Also** API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetNumOutputs`, `xPCGetStateLog`, `xPCGetTimeLog`

Target object method `xpctarget.xpc.getlog`

Target object property `TETLog`

# xPCGetTimeLog

---

**Purpose** Copy time log to array

**Prototype** `void xPCGetTimeLog(int port, int first_sample,  
int num_samples,  
int decimation, double *time_data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>first_sample</i>	Enter the index of the first sample to copy.
<i>num_samples</i>	Enter the number of samples to copy from the time log.
<i>decimation</i>	Select whether to copy all the sample values or every Nth value.
<i>time_data</i>	The log is stored in <i>time_data</i> , whose allocation is the responsibility of the caller.

**Description** The `xPCGetTimeLog` function gets the time log and copies the log into *time\_data*. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *first\_sample*, the sample indices range from 0 to (N-1), where N is the return value of `xPCNumLogSamples`. Use the `xPCNumLogSamples` function to get the number of samples.

Note that the target application must be stopped before you get the number.

**See Also** API functions `xPCNumLogWraps`, `xPCNumLogSamples`, `xPCMaxLogSamples`, `xPCGetStateLog`, `xPCGetTETLog`, `xPCSetLogMode`, `xPCGetLogMode`

Target object method `xpctarget.xpc.getlog`

Target object property `TimeLog`

<b>Purpose</b>	Initialize xPC Target DLL
<b>Prototype</b>	<code>int xPCInitAPI(void);</code>
<b>Return</b>	The xPCInitAPI function returns 0 if it completes execution without detecting an error. If the function detects an error, it returns -1.
<b>Description</b>	The xPCInitAPI function initializes the xPC Target dynamic link library. You must execute this function once at the beginning of the application to load the xPC Target API DLL. This function is defined in the file <code>xpcinitfree.c</code> . Link this file with your application.
<b>See Also</b>	API functions <code>xPCFreeAPI</code> , <code>xPCNumLogWraps</code> , <code>xPCNumLogSamples</code> , <code>xPCMaxLogSamples</code> , <code>xPCGetStateLog</code> , <code>xPCGetTETLog</code> , <code>xPCSetLogMode</code> , <code>xPCGetLogMode</code>

# xPCIsAppRunning

---

<b>Purpose</b>	Return target application running status
<b>Prototype</b>	<code>int xPCIsAppRunning(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	If the target application is stopped, the <code>xPCIsAppRunning</code> function returns 0. If the target application is running, this function returns 1. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCIsAppRunning</code> function returns 1 or 0 depending on whether the target application is stopped or running. If the function detects is an error, use the function <code>xPCGetLastError</code> to check for the error string constant.
<b>See Also</b>	API function <code>xPCIsOverloaded</code> Target object property <code>Status</code>

<b>Purpose</b>	Return target computer overload status
<b>Prototype</b>	<code>int xPCIsOverloaded(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	If the target application has overloaded the CPU, the <code>xPCIsOverloaded</code> function returns 1. If it has not overloaded the CPU, the function returns 0. If this function detects error, it returns -1.
<b>Description</b>	The <code>xPCIsOverloaded</code> function checks if the target application has overloaded the target computer and returns 1 if it has and 0 if it has not. If the target application is not running, the function returns 0.
<b>See Also</b>	API function <code>xPCIsAppRunning</code> Target object property <code>CPUOverload</code>

# xPCIsScFinished

---

**Purpose** Return data acquisition status for scope

**Prototype** `int xPCIsScFinished(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** If a scope finishes a data acquisition cycle, the `xPCIsScFinished` function returns 1. If the scope is in the process of acquiring data, this function returns 0. If the function detects an error, it returns -1.

**Description** The `xPCIsScFinished` function returns a Boolean value depending on whether scope *scNum* is finished (state of `SCST_FINISHED`) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state. Use the `xPCGetScope` function to get the scope number.

**See Also** API function `xPCScGetState`  
Scope object property `Status`



**Purpose** Load target application onto target computer

**Prototype** `void xPCLoadApp(int port, const char *pathstr, const char *filename);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>pathstr</i>	Enter the full path to the target application file, excluding the file name. For example, in C, use a string like "C:\\work".
<i>filename</i>	Enter the name of a compiled target application (*.dlm) without the file extension. For example, in C use a string like "xpcosc".

**Description** The `xPCLoadApp` function loads the compiled target application to the target computer. *pathstr* must not contain the trailing backslash. *pathstr* can be set to `NULL` or to the string 'nopath' if the application is in the current folder. The variable *filename* must not contain the target application extension.

Before returning, `xPCLoadApp` waits for a certain amount of time before checking whether the model initialization is complete. In the case where the model initialization is incomplete, `xPCLoadApp` returns a timeout error to indicate a connection problem (for example, `ETCPREAD`). By default, `xPCLoadApp` checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. The functions `xPCGetLoadTimeOut` and `xPCSetLoadTimeOut` control the number of attempts made.

# xPCLoadApp

---

## **See Also**

API functions `xPCStartApp`, `xPCStopApp`, `xPCUnloadApp`,  
`xPCSetLoadTimeOut`, `xPCGetLoadTimeOut`

Target object method `xpctarget.xpc.load`

**Purpose** Restore parameter values

**Prototype** `void xPCLoadParamSet(int port, const char *filename);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>filename</i>	Enter the name of the file that contains the saved parameters.

**Description** The `xPCLoadParamSet` function restores the target application parameter values saved in the file *filename*. This file must be located on a local drive of the target computer. The parameter file must have been saved from a previous call to `xPCSaveParamSet`.

**See Also** API function `xPCSaveParamSet`

# xPCMaxLogSamples

---

<b>Purpose</b>	Return maximum number of samples that can be in log buffer
<b>Prototype</b>	<code>int xPCMaxLogSamples(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCMaxLogSamples</code> function returns the total number of samples. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCMaxLogSamples</code> function returns the total number of samples that can be returned in the logging buffers.
<b>See Also</b>	API functions <code>xPCNumLogSamples</code> , <code>xPCNumLogWraps</code> , <code>xPCGetStateLog</code> , <code>xPCGetOutputLog</code> , <code>xPCGetTETLog</code> , <code>xPCGetTimeLog</code> Target object property <code>MaxLogSamples</code>

**Purpose** Copy maximum task execution time to array

**Prototype** `void xPCMaximumTET(int port, double *data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>data</i>	Array of at least two doubles.

**Description** The `xPCMaximumTET` function gets the maximum task execution time (TET) that was achieved during the previous target application run. This function also returns the time at which the maximum TET was achieved. The `xPCMaximumTET` function then copies these values into the *data* array. The maximum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

**See Also** API functions `xPCMinimumTET`, `xPCAverageTET`  
Target object property `MaxTET`

# xPCMinimumTET

---

**Purpose** Copy minimum task execution time to array

**Prototype** void xPCMinimumTET(int *port*, double \**data*);

**Arguments**

<i>port</i>	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
<i>data</i>	Array of at least two doubles.

**Description** The xPCMinimumTET function gets the minimum task execution time (TET) that was achieved during the previous target application run. This function also returns the time at which the minimum TET was achieved. The xPCMinimumTET function then copies these values into the *data* array. The minimum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.

**See Also** API functions xPCMaximumTET, xPCAverageTET  
Target object property MinTET

<b>Purpose</b>	Return number of samples in log buffer
<b>Prototype</b>	<code>int xPCNumLogSamples(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCNumLogSamples</code> function returns the number of samples in the log buffer. If the function detects an error, it returns -1.
<b>Description</b>	<p>The <code>xPCNumLogSamples</code> function returns the number of samples in the log buffer. In contrast to <code>xPCMaxLogSamples</code>, which returns the maximum number of samples that can be logged (because of buffer size constraints), <code>xPCNumLogSamples</code> returns the number of samples actually logged.</p> <p>Note that the target application must be stopped before you get the number.</p>
<b>See Also</b>	API functions <code>xPCGetStateLog</code> , <code>xPCGetOutputLog</code> , <code>xPCGetTETLog</code> , <code>xPCGetTimeLog</code> , <code>xPCMaxLogSamples</code>

# xPCNumLogWraps

---

<b>Purpose</b>	Return number of times log buffer wraps
<b>Prototype</b>	<code>int xPCNumLogWraps(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCNumLogWraps</code> function returns the number of times the log buffer wraps. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCNumLogWraps</code> function returns the number of times the log buffer wraps.
<b>See Also</b>	API functions <code>xPCNumLogSamples</code> , <code>xPCMaxLogSamples</code> , <code>xPCGetStateLog</code> , <code>xPCGetOutputLog</code> , <code>xPCGetTETLog</code> , <code>xPCGetTimeLog</code> Target object property <code>NumLogWraps</code>



**Purpose** Open connection to target computer

**Prototype** `void xPCOpenConnection(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCOpenConnection` function opens a connection to the target computer whose data is indexed by *port*. Before calling this function, set up the target information by calling `xPCRegisterTarget`. A call to either `xPCOpenSerialPort` or `xPCOpenTcpIpPort` can also set up the target information. If the port is already open, calling this function has no effect.

**See Also** API functions `xPCOpenTcpIpPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCTargetPing`, `xPCCloseConnection`, `xPCRegisterTarget`

# xPCOpenSerialPort

---

**Purpose** Open RS-232 connection to xPC Target system

**Prototype** `int xPCOpenSerialPort(int comPort, int baudRate);`

**Arguments**

<i>comPort</i>	Index of the COM port to be used (0 is COM1, 1 is COM2, and so forth).
<i>baudRate</i>	<i>baudRate</i> must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

**Return** The xPCOpenSerialPort function returns the port value for the connection. If the function detects an error, it returns -1.

**Description** The xPCOpenSerialPort function initiates an RS-232 connection to an xPC Target system. It returns the port value for the connection. Be sure to pass this value to all the xPC Target API functions that require a port value.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**See Also** API functions xPCOpenTcpIpPort, xPCClosePort, xPCReOpenPort, xPCTargetPing, xPCOpenConnection, xPCCloseConnection, xPCRegisterTarget, xPCDeRegisterTarget

<b>Purpose</b>	Open TCP/IP connection to xPC Target system				
<b>Prototype</b>	<pre>int xPCOpenTcpIpPort(const char *ipAddress, const char *ipPort);</pre>				
<b>Arguments</b>	<table><tr><td><i>ipAddress</i></td><td>Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".</td></tr><tr><td><i>ipPort</i></td><td>Enter the associated IP port as a string. For example, "22222".</td></tr></table>	<i>ipAddress</i>	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".	<i>ipPort</i>	Enter the associated IP port as a string. For example, "22222".
<i>ipAddress</i>	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".				
<i>ipPort</i>	Enter the associated IP port as a string. For example, "22222".				
<b>Return</b>	The xPCOpenTcpIpPort function returns a nonnegative integer that you can then use as the port value for an xPC Target API function that requires it. If this operation fails, this function returns -1.				
<b>Description</b>	The xPCOpenTcpIpPort function opens a connection to the TCP/IP location specified by the IP address. It returns a nonnegative integer if it succeeds. Use this integer as the <i>ipPort</i> variable in the xPC Target API functions that require a port value. The global error number is also set, which you can get using xPCGetLastError.				
<b>See Also</b>	API functions xPCOpenSerialPort, xPCClosePort, xPCReOpenPort, xPCTargetPing				

# xPCReboot

---

**Purpose** Reboot target computer

**Prototype** `void xPCReboot(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCReboot` function reboots the target computer. This function returns nothing. This function does not close the connection to the target computer. You should either explicitly close the port or call `xPCReOpenPort` once the target computer has rebooted.

**See Also** API function `xPCReOpenPort`  
Target object method `xpctarget.xpc.reboot`

<b>Purpose</b>	Reopen communication channel
<b>Prototype</b>	<code>int xPCReOpenPort(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCReOpenPort</code> function returns 0 if it reopens a connection without detecting an error. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCReOpenPort</code> function reopens the communications channel pointed to by <i>port</i> . The difference between this function and <code>xPCOpenSerialPort</code> or <code>xPCOpenTcpIpPort</code> is that <code>xPCReOpenPort</code> uses the already existing settings, while the other functions need to set up the port.
<b>See Also</b>	API functions <code>xPCOpenTcpIpPort</code> , <code>xPCClosePort</code>

# xPCRegisterTarget

---

**Purpose** Register target with xPC Target API library

**Prototype** `int xPCRegisterTarget(int commType, const char *ipAddress, const char *ipPort, int comPort, int baudRate);`

**Arguments** *commType* Specify the communication type (TCP/IP or RS-232) between the host and the target.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

*ipAddress* Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".

*ipPort* Enter the associated IP port as a string. For example, "22222".

*comPort* *comPort* and *baudRate* are as in xPCOpenSerialPort.

*baudRate* The *baudRate* must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

**Return** The xPCRegisterTarget function returns the port number. If the function detects an error, it returns -1.

**Description** The xPCRegisterTarget function works similarly to xPCOpenSerialPort and xPCOpenTcpIpPort, except that it does not try to open a connection to the target computer. In other words, xPCOpenSerialPort or xPCOpenTcpIpPort is equivalent to calling xPCRegisterTarget with the required parameters, followed by a call to xPCOpenConnection.

Use the constants COMMTYP\_TCP/IP and COMMTYP\_RS232 for *commType*. If *commType* is set to COMMTYP\_RS232, the function ignores *ipAddress*

and *ipPort*. Analogously, the function ignores *comPort* and *baudRate* if *commType* is set to `COMMTYP_TCPIP`.

If you enter a value of 0 for *baudRate*, this function sets the baud rate to the default value (115200).

### See Also

API functions `xPCDeRegisterTarget`, `xPCOpenTcpIpPort`, `xPCOpenSerialPort`, `xPCClosePort`, `xPCReOpenPort`, `xPCOpenConnection`, `xPCCloseConnection`, `xPCTargetPing`

# xPCRemScope

---

**Purpose** Remove scope

**Prototype** `void xPCRemScope(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Description** The `xPCRemScope` function removes the scope with number *scNum*. Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, see `xPCGetScopes`. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCAddScope`, `xPCScRemSignal`, `xPCGetScopes`  
Target object method `xpctarget.xpc.remscope`



**Purpose** Save parameter values of target application

**Prototype** `void xPCSaveParamSet(int port, const char *filename);`

**Arguments**

*port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*filename* Enter the name of the file to contain the saved parameters.

**Description** The `xPCSaveParamSet` function saves the target application parameter values in the file *filename*. This function saves the file on a local drive of the current target computer. You can later reload these parameters with the `xPCLoadParamSet` function.

You might want to save target application parameter values if you change these parameter values while the application is running in real time. Saving these values enable you to easily recreate target application parameter values from a number of application runs.

**See Also** API function `xPCLoadParamSet`

# xPCScAddSignal

---

**Purpose** Add signal to scope

**Prototype** `void xPCScAddSignal(int port, int scNum, int sigNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	Enter a signal number.

**Description** The `xPCScAddSignal` function adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScGetSignals` to get a list of the signals already present. Use the function `xPCGetScope` to get the scope number. Use the `xPCGetSignalIdx` function to get the signal number.

**See Also** API functions `xPCScRemSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`  
Scope object method `xpctarget.xpcsc.addsignal`

**Purpose** Scope autorestart status

**Prototype** `long xPCScGetAutoRestart(int port, int scNum)`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetAutoRestart` function returns the autorestart flag value of scope *scNum*. If the function detects an error, it returns -1.

**Description** The `xPCScGetAutoRestart` function gets the autorestart flag value for scope *scNum*. Autorestart flag can be disabled (0) or enabled (1).

**See Also** API functions `xPCScSetAutoRestart`

# xPCScGetData

---

**Purpose** Copy scope data to array

**Prototype** `void xPCScGetData(int port, int scNum, int signal_id, int start, int numsamples, int decimation, double *data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>signal_id</i>	Enter a signal number. Enter -1 to get time stamped data.
<i>start</i>	Enter the first sample from which data retrieval is to start
<i>numsamples</i>	Enter the number of samples retrieved with a decimation of <i>decimation</i> , starting from the <i>start</i> value.
<i>decimation</i>	Enter a value such that every <i>decimation</i> sample is retrieved in a scope window.
<i>data</i>	The data is available in the array <i>data</i> , starting from sample <i>start</i> .

**Description** The `xPCScGetData` function gets the data used in a scope. Use this function for scopes of type `SCTYPE_HOST`. The scope must be either in state "Finished" or in state "Interrupted" for the data to be retrievable. (Use the `xPCScGetState` function to check the state of the scope.) The data must be retrieved one signal at a time. The calling function must allocate the space ahead of time to store the scope data. *data* must be an array of doubles, regardless of the data type of the signal to be retrieved. Use the function `xPCScGetSignals` to get the list of signals in the scope for *signal\_id*. Use the function `xPCGetScope` to get the scope number for *scNum*.

To get time stamped data, specify -1 for `signal_id`. From the output, you can then get the number of nonzero elements.

### **See Also**

API functions `xPCGetScope`, `xPCScGetState`, `xPCScGetSignals`

Scope object property `Data`

# xPCScGetDecimation

---

**Purpose** Return decimation of scope

**Prototype** `int xPCScGetDecimation(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetDecimation` function returns the decimation of scope *scNum*. If the function detects an error, it returns -1.

**Description** The `xPCScGetDecimation` function gets the decimation of scope *scNum*. The decimation is a number, N, meaning every Nth sample is acquired in a scope window. Use the `xPCGetScope` function to get the scope number.

**See Also** API function `xPCScSetDecimation`  
Scope object property `Decimation`

**Purpose** Get number of pre- or post-triggering samples before triggering scope

**Prototype** `int xPCScGetNumPrePostSamples(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetNumPrePostSamples` function returns the number of samples for pre- or posttriggering for scope *scNum*. If an error occurs, this function returns the minimum integer value (-2147483647-1).

**Description** The `xPCScGetNumPrePostSamples` function gets the number of samples for pre- or posttriggering for scope *scNum*. A negative number implies pretriggering, whereas a positive number implies posttriggering samples. Use the `xPCGetScope` function to get the scope number.

**See Also** API function `xPCScSetNumPrePostSamples`  
Scope object property `NumPrePostSamples`

# xPCScGetNumSamples

---

**Purpose** Get number of samples in one data acquisition cycle

**Prototype** `int xPCScGetNumSamples(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetNumSamples` function returns the number of samples in the scope *scNum*. If the function detects an error, it returns -1.

**Description** The `xPCScGetNumSamples` function gets the number of samples in one data acquisition cycle for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

**See Also** API function `xPCScSetNumSamples`  
Scope object property `NumSamples`



<b>Purpose</b>	Get number of signals in scope				
<b>Prototype</b>	<code>int xPCScGetNumSignals(int <i>port</i>, int <i>scNum</i>);</code>				
<b>Arguments</b>	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>scNum</i></td><td>Enter the scope number.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>scNum</i>	Enter the scope number.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>scNum</i>	Enter the scope number.				
<b>Return</b>	The <code>xPCScGetNumSignals</code> function returns the number of signals in the scope <i>scNum</i> . If the function detects an error, it returns -1.				
<b>Description</b>	The <code>xPCScGetNumSignals</code> function gets the number of signals in the scope <i>scNum</i> . Use the <code>xPCGetScope</code> function to get the scope number.				
<b>See Also</b>	API function <code>xPCGetScope</code>				

# xPCScGetSignalList

---

**Purpose** Copy list of signals to array

**Prototype** `void xPCScGetSignalList(int port, int scNum, int *data)`

**Arguments**

<i>port</i>	Value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>data</i>	Integer array allocated by the caller as a list containing the signal identifiers.

**Description** The `xPCScGetSignals` function gets the list of signals defined for scope *scNum*. The array *data* must be large enough to hold the list of signals. To query the size, use the `xPCScGetNumSignals` function. Use the `xPCGetScope` function to get the scope number.

---

**Note** Use the `xPCScGetSignalList` function instead of the `xPCScGetSignals` function. The `xPCScGetSignals` will be obsoleted in a future release.

---

**Purpose** Copy list of signals to array

**Prototype** `void xPCScGetSignals(int port, int scNum, int *data);`

**Arguments**

<i>port</i>	Value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>data</i>	Integer array allocated by the caller as a list containing the signal identifiers, terminated by -1.

**Description** The `xPCScGetSignals` function gets the list of signals defined for scope *scNum*. You can use the constant `MAX_SIGNALS`, defined in `xpcapiconst.h`, as the size of *data*. Use the `xPCGetScope` function to get the scope number.

---

**Note** This function will be obsoleted in a future release. Use the `xPCScGetSignalList` function instead.

---

**See Also** API functions `xPCScGetData`, `xPCGetScopes`  
Scope object property `Signals`

# xPCScGetStartTime

---

**Purpose** Get start time for last data acquisition cycle

**Prototype** `double xPCScGetStartTime(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetStartTime` function returns the start time for the last data acquisition cycle of a scope. If the function detects an error, it returns -1.

**Description** The `xPCScGetStartTime` function gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type `SCTYPE_HOST`. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCScGetNumSamples`, `xPCScGetDecimation`

**Purpose** Get state of scope

**Prototype** `int xPCScGetState(int port, int scNum);`

**Arguments**

*port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum* Enter the scope number.

**Return** The `xPCScGetState` function returns the state of scope *scNum*. If the function detects an error, it returns -1.

**Description** The `xPCScGetState` function gets the state of scope *scNum*, or -1 upon error. Use the `xPCGetScope` function to get the scope number.

Constants to find the scope state, defined in `xpcapiconst.h`, have the following meanings:

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.

## xPCScGetState

---

Constant	Value	Description
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

### See Also

API functions `xPCScStart`, `xPCScStop`

Scope object property `Status`

**Purpose** Get trigger level for scope

**Prototype** `double xPCScGetTriggerLevel(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetTriggerLevel` function returns the scope trigger level. If the function detects an error, it returns -1.

**Description** The `xPCScGetTriggerLevel` function gets the trigger level for scope *scNum*. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCScSetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`  
Scope object property `TriggerLevel`

# xPCScGetTriggerMode

---

**Purpose** Get trigger mode for scope

**Prototype** `int xPCScGetTriggerMode(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetTriggerMode` function returns the scope trigger mode. If the function detects an error, it returns -1.

**Description** The `xPCScGetTriggerMode` function gets the trigger mode for scope *scNum*. Use the `xPCGetScope` function to get the scope number. Use the constants defined in `xpcapiconst.h` to interpret the trigger mode. These constants include the following:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.



Constant	Value	Description
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

## See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`

Scope object method `trigger`

Scope object property `TriggerMode`

# xPCScGetTriggerScope

---

<b>Purpose</b>	Get trigger scope				
<b>Prototype</b>	<code>int xPCScGetTriggerScope(int <i>port</i>, int <i>scNum</i>);</code>				
<b>Arguments</b>	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr><tr><td><i>scNum</i></td><td>Enter the scope number.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .	<i>scNum</i>	Enter the scope number.
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .				
<i>scNum</i>	Enter the scope number.				
<b>Return</b>	The <code>xPCScGetTriggerScope</code> function returns a trigger scope. If the function detects an error, it returns -1.				
<b>Description</b>	The <code>xPCScGetTriggerScope</code> function gets the trigger scope for scope <i>scNum</i> . Use the <code>xPCGetScope</code> function to get the scope number.				
<b>See Also</b>	API functions <code>xPCScSetTriggerLevel</code> , <code>xPCScGetTriggerLevel</code> , <code>xPCScSetTriggerSlope</code> , <code>xPCScGetTriggerSlope</code> , <code>xPCScSetTriggerSignal</code> , <code>xPCScGetTriggerSignal</code> , <code>xPCScSetTriggerMode</code> , <code>xPCScGetTriggerMode</code> Scope object property <code>TriggerScope</code>				

**Purpose** Get sample number for triggering scope

**Prototype** `int xPCScGetTriggerScopeSample(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetTriggerScopeSample` function returns a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the function detects an error, it returns `INT_MIN` (-2147483647-1).

**Description** The `xPCScGetTriggerScopeSample` function gets the number of samples a triggering scope (*scNum*) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

**See Also** API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScSetTriggerScopeSample`

Scope object property `TriggerSample`

# xPCScGetTriggerSignal

---

**Purpose** Get trigger signal for scope

**Prototype** `int xPCScGetTriggerSignal(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetTriggerSignal` function returns the scope trigger signal. If the function detects an error, it returns -1.

**Description** The `xPCScGetTriggerSignal` function gets the trigger signal for scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope.

**See Also** API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object method `trigger`

Scope object property `TriggerSignal`

**Purpose** Get trigger slope for scope

**Prototype** `int xPCScGetTriggerSlope(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetTriggerSlope` function returns the scope trigger slope. If the function detects an error, it returns -1.

**Description** The `xPCScGetTriggerSlope` function gets the trigger slope of scope *scNum*. Use the `xPCGetScope` function to get the scope number for the trigger scope. Use the constants defined in `xpcapiconst.h` to interpret the trigger slope. These constants have the following meanings:

Constant	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger slope must be rising when the signal crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger slope must be falling when the signal crosses the trigger value.

# xPCScGetTriggerSlope

---

## See Also

API functions `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`,  
`xPCScSetTriggerSlope`, `xPCScSetTriggerSignal`,  
`xPCScGetTriggerSignal`, `xPCScSetTriggerScope`,  
`xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`

Scope object method `xpctarget.xpcsc.trigger`

Scope object properties `TriggerMode`, `TriggerSlope`

**Purpose** Get type of scope

**Prototype** `int xPCScGetType(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCScGetType` function returns the scope type. If the function detects an error, it returns -1.

**Description** The `xPCScGetType` function gets the type (`SCTYPE_HOST` for host, `SCTYPE_TARGET` for target, or `SCTYPE_FILE` for file) of scope *scNum*. Use the constants defined in `xpcapiconst.h` to interpret the return value. A scope of type `SCTYPE_HOST` is displayed on the host computer while a scope of type `SCTYPE_TARGET` is displayed on the target computer screen. A scope of type `SCTYPE_FILE` is stored on a storage medium. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCAddScope`, `xPCRemScope`  
Scope object property `Type`

# xPCScRemSignal

---

**Purpose** Remove signal from scope

**Prototype** `void xPCScRemSignal(int port, int scNum, int sigNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>sigNum</i>	Enter a signal number.

**Description** The `xPCScRemSignal` function removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCGetScopes` to determine the existing scopes, and use `xPCScGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCScAddSignal`, `xPCAddScope`, `xPCRemScope`, `xPCGetScopes`, `xPCScGetSignals`, `xPCScGetState`  
Scope object method `remsignal`



**Purpose** Scope autorestart status

**Prototype** `void xPCScSetAutoRestart(int port, int scNum, int autorestart)`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>autorestart</i>	Enter value to enable (1) or disable (0) scope autorestart.

**Description** The `xPCScSetAutoRestart` function sets the autorestart flag for scope *scNum* to 0 or 1. 0 disables the flag, 1 enables it. Use this function only when the scope is stopped.

**See Also** API functions `xPCScGetAutoRestart`

# xPCScSetDecimation

---

**Purpose** Set decimation of scope

**Prototype** `void xPCScSetDecimation(int port, int scNum, int decimation);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>decimation</i>	Enter an integer for the decimation.

**Description** The `xPCScSetDecimation` function sets the *decimation* of scope *scNum*. The decimation is a number, N, meaning every Nth sample is acquired in a scope window. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCScGetDecimation`, `xPCScGetState`  
Scope object property `Decimation`

**Purpose** Set number of pre- or posttriggering samples before triggering scope

**Prototype** `void xPCScSetNumPrePostSamples(int port, int scNum, int prepost);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>prepost</i>	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

**Description** The `xPCScSetNumPrePostSamples` function sets the number of samples for pre- or posttriggering for scope *scNum* to *prepost*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCScGetNumPrePostSamples`, `xPCScGetState`  
Scope object property `NumPrePostSamples`

# xPCScSetNumSamples

---

**Purpose** Set number of samples in one data acquisition cycle

**Prototype** `void xPCScSetNumSamples(int port, int scNum, int samples);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>samples</i>	Enter the number of samples you want to acquire in one cycle.

**Description** The `xPCScSetNumSamples` function sets the number of samples for scope *scNum* to *samples*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number.

**See Also** API functions `xPCScGetNumSamples`, `xPCScGetState`  
Scope object property `NumSamples`

**Purpose** Set trigger level for scope

**Prototype** `void xPCScSetTriggerLevel(int port, int scNum, double level);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>level</i>	Value for a signal to trigger data acquisition with a scope.

**Description** The `xPCScSetTriggerLevel` function sets the trigger level to *level* for scope *scNum*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScope` function to get the scope number for the trigger scope.

**See Also** API functions `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Scope object property `TriggerLevel`

# xPCScSetTriggerMode

---

**Purpose** Set trigger mode of scope

**Prototype** `void xPCScSetTriggerMode(int port, int scNum, int mode);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	Trigger mode for a scope.

**Description** The `xPCScSetTriggerMode` function sets the trigger mode of scope *scNum* to *mode*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances. This is the default.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

## See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScSetTriggerSlope, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScGetTriggerMode, xPCScGetState

Scope object method trigger

Scope object property TriggerMode

# xPCScSetTriggerScope

---

**Purpose** Select scope to trigger another scope

**Prototype** `void xPCScSetTriggerScope(int port, int scNum, int trigScope);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigScope</i>	Enter the scope number of the scope used for a trigger.

**Description** The `xPCScSetTriggerScope` function sets the trigger scope of scope *scNum* to *trigScope*. This function can only be used when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

The scope type can be `SCTYPE_HOST`, `SCTYPE_TARGET`, or `SCTYPE_FILE`.

**See Also** API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetState`

Scope object property `TriggerScope`



## Purpose

Set sample number for triggering scope

## Prototype

```
void xPCScSetTriggerScopeSample(int port, int scNum, int trigScSamp);
```

## Arguments

*port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum* Enter the scope number.

*trigScSamp* Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.

## Description

The `xPCScSetTriggerScopeSample` function sets the number of samples (*trigScSamp*) a triggering scope acquires before it triggers a second scope (*scNum*). Use the `xPCGetScopes` function to get a list of scopes.

For meaningful results, set *trigScSamp* between -1 and (*nSamp*-1). *nSamp* is the number of samples in one data acquisition cycle for the triggering scope. If you specify too large a value, the scope is never triggered.

If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, enter a value of -1 for *trigScSamp*.

## See Also

API functions `xPCGetScopes`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScSetTriggerSignal`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`, `xPCScGetTriggerScopeSample`

Scope object properties `TriggerMode`, `TriggerSample`

# xPCScSetTriggerSignal

---

**Purpose** Select signal to trigger scope

**Prototype** `void xPCScSetTriggerSignal(int port, int scNum, int trigSig);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSig</i>	Enter a signal number.

**Description** The `xPCScSetTriggerSignal` function sets the trigger signal of scope *scNum* to *trigSig*. The trigger signal *trigSig* must be one of the signals in the scope. Use this function only when the scope is stopped. You can use `xPCScGetSignals` to get the list of signals in the scope. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCScGetState`, `xPCScSetTriggerLevel`, `xPCScGetTriggerLevel`, `xPCScSetTriggerSlope`, `xPCScGetTriggerSlope`, `xPCScGetTriggerSignal`, `xPCScSetTriggerScope`, `xPCScGetTriggerScope`, `xPCScSetTriggerMode`, `xPCScGetTriggerMode`  
Scope object property `TriggerSignal`

**Purpose** Set slope of signal that triggers scope

**Prototype** `void xPCScSetTriggerSlope(int port, int scNum, int trigSlope);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>trigSlope</i>	Enter the slope mode for the signal that triggers the scope.

**Description** The `xPCScSetTriggerSlope` function sets the trigger slope of scope *scNum* to *trigSlope*. Use this function only when the scope is stopped. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

Use the constants defined in `xpcapiconst.h` to set the trigger slope:

Constant	Value	Description
<code>TRIGSLOPE_EITHER</code>	0	The trigger slope can be either rising or falling.
<code>TRIGSLOPE_RISING</code>	1	The trigger signal value must be rising when it crosses the trigger value.
<code>TRIGSLOPE_FALLING</code>	2	The trigger signal value must be falling when it crosses the trigger value.

# xPCScSetTriggerSlope

---

## See Also

API functions xPCGetScopes, xPCScSetTriggerLevel, xPCScGetTriggerLevel, xPCScGetTriggerSlope, xPCScSetTriggerSignal, xPCScGetTriggerSignal, xPCScSetTriggerScope, xPCScGetTriggerScope, xPCScSetTriggerMode, xPCScGetTriggerMode, xPCScGetState

Scope object property TriggerSlope

**Purpose** Set software trigger of scope

**Prototype** `void xPCScSoftwareTrigger(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Description** The `xPCScSoftwareTrigger` function triggers scope *scNum*. The scope must be in the state `Waiting for trigger` for this function to succeed. Use `xPCScGetState` to check the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

You can use the `xPCScSoftwareTrigger` function to trigger the scope, regardless of the trigger mode.

**See Also** API functions `xPCGetScopes`, `xPCScGetState`, `xPCIsScFinished`  
Scope object method `trigger`  
Scope object property `TriggerMode`

# xPCScStart

---

**Purpose** Start data acquisition for scope

**Prototype** `void xPCScStart(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Description** The `xPCScStart` function starts or restarts the data acquisition of scope *scNum*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. Call `xPCScGetState` to check the state of the scope or, for host scopes that are already started, call `xPCIsScFinished`. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCScGetState`, `xPCScStop`, `xPCIsScFinished`  
Scope object method `start` (scope object)

**Purpose** Stop data acquisition for scope

**Prototype** `void xPCScStop(int port, int scNum);`

**Arguments**

*port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum* Enter the scope number.

**Description** The `xPCScStop` function stops the scope *scNum*. This sets the scope to the "Interrupted" state. The scope must be running for this function to succeed. Use `xPCScGetState` to determine the state of the scope. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCScStart`, `xPCScGetState`  
Scope object method `stop` (scope object)

# xPCSetEcho

---

**Purpose** Turn message display on or off

**Prototype** `void xPCSetEcho(int port, int mode);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>mode</i>	Valid values are
0	Turns the display off
1	Turns the display on

**Description** On the target computer screen, the `xPCSetEcho` function sets the message display on the target computer on or off. You can change the mode only when the target application is stopped. When you turn the message display off, the message screen no longer updates.

**See Also** API function `xPCGetEcho`



**Purpose** Set last error to specific string constant

**Prototype** `void xPCSetLastError(int error);`

**Arguments** *error* Specify the string constant for the error.

**Description** The xPCSetLastError function sets the global error constant returned by xPCGetLastError to *error*. This is useful only to set the string constant to ENOERR, indicating no error was found.

**See Also** API functions xPCGetLastError, xPCErrorMsg

# xPCSetLoadTimeOut

---

**Purpose** Change initialization timeout value between host computer and target computer

**Prototype** `void xPCSetLoadTimeOut(int port, int timeOut);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>timeOut</i>	Enter the new communication timeout value.

**Description** The `xPCSetLoadTimeOut` function changes the timeout value for communication between the host computer and target computer. The *timeOut* value is the time an xPC Target API function waits for the communication between the host computer and target computer to complete before returning. It enables you to set the number of communication attempts to be made before signaling a timeout.

For example, the function `xPCLoadApp` waits to check whether the model initialization for a new application is complete before returning. When a new target application is loaded onto the target computer, the function `xPCLoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCLoadApp` returns a timeout error.

By default, `xPCLoadApp` checks for target readiness for up to 5 seconds. However, for larger models or models requiring longer initialization (for example, models with thermocouple boards), the default might not be long enough and a spurious timeout can be generated. Other functions that communicate with the target computer will wait for *timeOut* seconds before declaring a timeout event.

**See Also** API functions `xPCGetLoadTimeOut`, `xPCLoadApp`, `xPCUnloadApp`

**Purpose** Set logging mode and increment value of scope

**Prototype** `void xPCSetLogMode(int port, lgmode logging_data);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>logging_data</i>	Logging mode and increment value.

**Description** The `xPCSetLogMode` function sets the logging mode and increment to the values set in *logging\_data*. See the structure `lgmode` for more details.

**See Also**

- API function `xPCGetLogMode`
- API structure `lgmode`
- Target object property `LogMode`

# xPCSetParam

---

**Purpose** Change value of parameter

**Prototype** `void xPCSetParam(int port, int paramIdx, const double *paramValue);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>paramIdx</i>	Parameter index.
<i>paramValue</i>	Vector of doubles, assumed to be the size required by the parameter type

**Description** The `xPCSetParam` function sets the parameter *paramIdx* to the value in *paramValue*. For matrices, *paramValue* should be a vector representation of the matrix in column-major format. Although *paramValue* is a vector of doubles, the function converts the values to the expected data types (using truncation) before setting them.

**See Also** API functions `xPCGetParamDims`, `xPCGetParamIdx`, `xPCGetParam`

**Purpose** Change target application sample time

**Prototype** `void xPCSetSampleTime(int port, double ts);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>ts</i>	Sample time for the target application.

**Description** The `xPCSetSampleTime` function sets the sample time, in seconds, of the target application to *ts*. Use this function only when the application is stopped.

**See Also** API function `xPCGetSampleTime`  
Target object property `SampleTime`

# xPCSetScope

---

**Purpose** Set properties of scope

**Prototype** `void xPCSetScope(int port, scopedata state);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>state</i>	Enter a structure of type <code>scopedata</code> .

**Description** The `xPCSetScope` function sets the properties of a scope using a *state* structure of type `scopedata`. Set the properties you want to set for the scope. You can set several properties at the same time. For convenience, call the function `xPCGetScope` first to populate the structure with the current values. You can then change the desired values. Use this function only when the scope is stopped. Use `xPCScGetState` to determine the state of the scope.

**See Also** API functions `xPCGetScope`, `xPCScGetState`, `scopedata`  
Scope object method `set (scope object)`

**Purpose** Change target application stop time

**Prototype** `void xPCSetStopTime(int port, double tfinal);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>tfinal</i>	Enter the stop time, in seconds.

**Description** The `xPCSetStopTime` function sets the stop time of the target application to the value in *tfinal*. The target application will run for this number of seconds before stopping. Set *tfinal* to `-1.0` to set the stop time to infinity.

**See Also** API function `xPCGetStopTime`  
Target object property `StopTime`

# xPCStartApp

---

**Purpose** Start target application

**Prototype** `void xPCStartApp(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCStartApp` function starts the target application loaded on the target machine.

**See Also** API function `xPCStopApp`  
Target object method `start` (target application object)



**Purpose** Stop target application

**Prototype** void xPCStopApp(int *port*);

**Arguments** *port* Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.

**Description** The xPCStopApp function stops the target application loaded on the target computer. The target application remains loaded and the parameter changes you made remain intact. If you want to stop and unload an application, use xPCUnloadApp.

**See Also** API functions xPCStartApp, xPCUnloadApp  
Target object method stop (target application object)

# xPCTargetPing

---

<b>Purpose</b>	Ping target computer		
<b>Prototype</b>	<code>int xPCTargetPing(int <i>port</i>);</code>		
<b>Arguments</b>	<table><tr><td><i>port</i></td><td>Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code>.</td></tr></table>	<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .		
<b>Return</b>	The <code>xPCTargetPing</code> function does not return an error status. This function returns 1 if the target responds. If the target computer does not respond, the function returns 0.		
<b>Description</b>	<p>The <code>xPCTargetPing</code> function pings the target computer and returns 1 or 0 depending on whether the target responds or not. This function returns an error string constant only when there is an error in the input parameter (for example, the port number is invalid or <i>port</i> is not open). Other errors, such as the inability to connect to the target, are ignored.</p> <p>If you are using TCP/IP, note that <code>xPCTargetPing</code> will cause the target computer to close the TCP/IP connection. You can use <code>xPCOpenConnection</code> to reconnect. You can also use this <code>xPCTargetPing</code> feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if your host side program crashes).</p>		
<b>See Also</b>	API functions <code>xPCOpenConnection</code> , <code>xPCOpenSerialPort</code> , <code>xPCOpenTcpIpPort</code> , <code>xPCClosePort</code>		

**Purpose** Get status of grid line for particular scope

**Prototype** `int xPCTgScGetGrid(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** Returns the status of the grid for a scope of type `SCTYPE_TARGET`. If the function detects an error, it returns -1.

**Description** The `xPCTgScGetGrid` function gets the state of the grid lines for scope *scNum* (which must be of type `SCTYPE_TARGET`). A return value of 1 implies grid on, while 0 implies grid off. Note that when the scope mode is set to `SCMODE_NUMERICAL`, the grid is not drawn even when the grid mode is set to 1.

---

### Tip

- Use `xPCTgScSetMode` and `xPCTgScGetMode` to set and retrieve the scope mode.
  - Use `xPCGetScopes` to get a list of scopes.
- 

**See Also** API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

# xPCTgScGetMode

---

**Purpose** Get scope mode for displaying signals

**Prototype** `int xPCTgScGetMode(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Return** The `xPCTgScGetMode` function returns the value corresponding to the scope mode. The possible values are

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

If this function detects an error, it returns -1.

**Description** The `xPCTgScGetMode` function gets the mode (`SCMODE_NUMERICAL`, `SCMODE_REDRAW`, `SCMODE_SLIDING`, `SCMODE_ROLLING`) of the scope *scNum*, which must be of type `SCTYPE_TARGET`. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Mode`

<b>Purpose</b>	Get view mode for target computer display
<b>Prototype</b>	<code>int xPCTgScGetViewMode(int <i>port</i>);</code>
<b>Arguments</b>	<i>port</i> Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<b>Return</b>	The <code>xPCTgScGetViewMode</code> function returns the view mode for the target computer screen. If the function detects an error, it returns -1.
<b>Description</b>	The <code>xPCTgScGetViewMode</code> function gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is that of the scope currently displayed on the screen. If the value is 0, then all defined scopes are displayed on the target computer screen, but no scopes are in focus (all scopes are unzoomed).
<b>See Also</b>	API functions <code>xPCGetScopes</code> , <code>xPCTgScSetGrid</code> , <code>xPCTgScGetGrid</code> , <code>xPCTgScSetViewMode</code> , <code>xPCTgScSetMode</code> , <code>xPCTgScGetMode</code> , <code>xPCTgScSetYLimits</code> , <code>xPCTgScGetYLimits</code> Target object property <code>ViewMode</code>

# xPCTgScGetYLimits

---

**Purpose** Copy *y*-axis limits for scope to array

**Prototype** `void xPCTgScGetYLimits(int port, int scNum, double *limits);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>limits</i>	The first element of the array is the lower limit while the second element is the upper limit.

**Description** The `xPCTgScGetYLimits` function gets and copies the upper and lower limits for a scope of type `SCTYPE_TARGET` and with scope number *scNum*. The limits are stored in the array *limits*. If both elements are zero, the limits are autoscaled. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`

Scope object property `YLimit`

**Purpose** Set grid mode for scope

**Prototype** void xPCTgScSetGrid(int *port*, int *scNum*, int *grid*);

**Arguments**

<i>port</i>	Enter the value returned by either the function xPCOpenSerialPort or the function xPCOpenTcpIpPort.
<i>scNum</i>	Enter the scope number.
<i>grid</i>	Enter a grid value.

**Description** The xPCTgScSetGrid function sets the grid of a scope of type SCTYPE\_TARGET and scope number *scNum* to *grid*. If *grid* is 0, the grid is off. If *grid* is 1, the grid is on and grid lines are drawn on the scope window. When the drawing mode of scope *scNum* is set to SCMODE\_NUMERICAL, the grid is not drawn even when the grid mode is set to 1. Use the xPCGetScopes function to get a list of scopes.

**See Also** API functions xPCGetScopes, xPCTgScGetGrid, xPCTgScSetViewMode, xPCTgScGetViewMode, xPCTgScSetMode, xPCTgScGetMode, xPCTgScSetYLimits, xPCTgScGetYLimits

Scope object property Grid

# xPCTgScSetMode

---

**Purpose** Set display mode for scope

**Prototype** `void xPCTgScSetMode(int port, int scNum, int mode);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.
<i>mode</i>	Enter the value for the mode.

**Description** The `xPCTgScSetMode` function sets the mode of a scope of type `SCTYPE_TARGET` and scope number *scNum* to *mode*. You can use one of the following constants for *mode*:

- `SCMODE_NUMERICAL = 0`
- `SCMODE_REDRAW = 1`
- `SCMODE_SLIDING = 2`
- `SCMODE_ROLLING = 3`

Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`

Scope object property `Mode`



**Purpose** Set view mode for scope

**Prototype** `void xPCTgScSetViewMode(int port, int scNum);`

**Arguments**

<i>port</i>	Enter the value returned by either the function <code>xPCOpenSerialPort</code> or the function <code>xPCOpenTcpIpPort</code> .
<i>scNum</i>	Enter the scope number.

**Description** The `xPCTgScSetViewMode` function sets the target computer screen to display one scope with scope number *scNum*. If you set *scNum* to 0, the target computer screen displays all the defined scopes. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScSetYLimits`, `xPCTgScGetYLimits`  
Target object property `ViewMode`

# xPCTgScSetYLimits

---

**Purpose** Set *y*-axis limits for scope

**Prototype** `void xPCTgScSetYLimits(int port, int scNum, const double *Ylimits);`

**Arguments**

*port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

*scNum* Enter the scope number.

*Ylimits* Enter a two-element array.

**Description** The `xPCTgScSetYLimits` function sets the *y*-axis limits for a scope with scope number *scNum* and type `SCTYPE_TARGET` to the values in the double array *Ylimits*. The first element is the lower limit, and the second element is the upper limit. Set both limits to 0.0 to specify autoscaling. Use the `xPCGetScopes` function to get a list of scopes.

**See Also** API functions `xPCGetScopes`, `xPCTgScSetGrid`, `xPCTgScGetGrid`, `xPCTgScSetViewMode`, `xPCTgScGetViewMode`, `xPCTgScSetMode`, `xPCTgScGetMode`, `xPCTgScGetYLimits`

Scope object property `Ylimit`

**Purpose** Unload target application

**Prototype** `void xPCUnloadApp(int port);`

**Arguments** *port* Enter the value returned by either the function `xPCOpenSerialPort` or the function `xPCOpenTcpIpPort`.

**Description** The `xPCUnloadApp` function stops the current target application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new target application. The function `xPCLoadApp` calls this function before loading a new target application.

**See Also** API function `xPCLoadApp`  
Target object methods `xpctarget.xpc.load`, `xpctarget.xpc.unload`

# xPCUnloadApp

---

# xPC Target API Reference for COM

---

- “COM API Methods” on page 3-2
- “COM API Methods — Alphabetical List” on page 3-9

## COM API Methods

In this section...
“Target Computers” on page 3-2
“Target Applications” on page 3-3
“Scopes” on page 3-4
“Parameters” on page 3-6
“Signals” on page 3-6
“Data Logs” on page 3-7
“File Systems” on page 3-7
“Errors” on page 3-8

---

**Note** The xPC Target COM API is no longer being enhanced. You should use the xPC Target API for Microsoft .NET Framework instead. See “xPC Target API for Microsoft .NET Framework”

---

### Target Computers

xPCProtocol.Close	Close RS-232 or TCP/IP communication connection
xPCProtocol.GetLoadTimeOut	Return current timeout value for target application initialization
xPCProtocol.Init	Initialize xPC Target API DLL
xPCProtocol.Port	Contain communication channel index
xPCProtocol.Reboot	Reboot target computer
xPCProtocol.RS232Connect	Open RS-232 connection to target computer
xPCProtocol.SetLoadTimeOut	Change initialization timeout value
xPCProtocol.TargetPing	Ping target computer

xPCProtocol.TcpIpConnect	Open TCP/IP connection to target computer
xPCProtocol.Term	Unload xPC Target API DLL from memory
xPCTarget.UnLoadApp	Unload target application

## Target Applications

xPCTarget.AverageTET	Get average task execution time
xPCTarget.GetAppName	Get target application name
xPCTarget.GetExecTime	Get execution time for target application
xPCTarget.GetNumOutputs	Get number of outputs
xPCTarget.GetSampleTime	Get sample time
xPCTarget.GetStopTime	Get stop time
xPCTarget.Init	Initialize target object to communicate with target computer
xPCTarget.IsAppRunning	Return running status for target application
xPCTarget.IsOverloaded	Return overload status for target computer
xPCTarget.MaximumTET	Copy maximum task execution time to array
xPCTarget.MaxLogSamples	Return maximum number of samples that can be in log buffer
xPCTarget.MinimumTET	Copy minimum task execution time to array
xPCTarget.NumLogSamples	Return number of samples in log buffer
xPCTarget.NumLogWraps	Return number of times log buffer wraps

xPCTarget.SetSampleTime	Change sample time for target application
xPCTarget.SetStopTime	Change stop time of target application
xPCTarget.StartApp	Start target application
xPCTarget.StopApp	Stop target application

## Scopes

xPCScopes.AddFileScope	Create new file scope
xPCScopes.AddHostScope	Create new host scope
xPCScopes.AddTargetScope	Create new target scope
xPCScopes.GetScopes	Get and copy list of scope numbers
xPCScopes.Init	Initialize scope object to communicate with target computer
xPCScopes.IsScopeFinished	Get data acquisition status for scope
xPCScopes.RemScope	Remove scope
xPCScopes.ScopeAddSignal	Add signal to scope
xPCScopes.ScopeGetAutoRestart	Scope autorestart value
xPCScopes.ScopeGetData	Copy scope data to array
xPCScopes.ScopeGetDecimation	Get decimation of scope
xPCScopes.ScopeGetNumPrePostSamples	Get number of pre- or posttriggering samples before triggering scope
xPCScopes.ScopeGetNumSamples	Get number of samples in one data acquisition cycle
xPCScopes.ScopeGetSignals	Get list of signals
xPCScopes.ScopeGetStartTime	Get last data acquisition cycle start time
xPCScopes.ScopeGetState	Get state of scope
xPCScopes.ScopeGetTriggerLevel	Get trigger level for scope



xPCScopes.ScopeGetTriggerMode	Get trigger mode for scope
xPCScopes.ScopeGetTriggerModeStr	Get trigger mode as string
xPCScopes.ScopeGetTriggerSample	Get sample number for triggering scope
xPCScopes.ScopeGetTriggerSignal	Get trigger signal for scope
xPCScopes.ScopeGetTriggerSlope	Get trigger slope for scope
xPCScopes.ScopeGetTriggerSlopeStr	Get trigger slope as string
xPCScopes.ScopeGetType	Get type of scope
xPCScopes.ScopeRemSignal	Remove signal from scope
xPCScopes.ScopeSetAutoRestart	Scope autorestart value
xPCScopes.ScopeSetDecimation	Set decimation of scope
xPCScopes.ScopeSetNumPrePostSamples	Set number of pre- or posttriggering samples before triggering scope
xPCScopes.ScopeSetNumSamples	Set number of samples in one data acquisition cycle
xPCScopes.ScopeSetTriggerLevel	Set trigger level for scope
xPCScopes.ScopeSetTriggerMode	Set trigger mode of scope
xPCScopes.ScopeSetTriggerSample	Set sample number for triggering scope
xPCScopes.ScopeSetTriggerSignal	Select signal to trigger scope
xPCScopes.ScopeSetTriggerSlope	Set slope of signal that triggers scope
xPCScopes.ScopeSoftwareTrigger	Set software trigger of scope
xPCScopes.ScopeStart	Start data acquisition for scope
xPCScopes.ScopeStop	Stop data acquisition for scope
xPCScopes.TargetScopeGetGrid	Get status of grid line for particular scope
xPCScopes.TargetScopeGetMode	Get scope mode for displaying signals

xPCScopes.TargetScopeGetModeStr	Get scope mode string for displaying signals
xPCScopes.TargetScopeGetViewMode	Get view mode for target computer display
xPCScopes.TargetScopeGetYLimits	Get y-axis limits for scope
xPCScopes.TargetScopeSetGrid	Set grid mode for scope
xPCScopes.TargetScopeSetMode	Set display mode for scope
xPCScopes.TargetScopeSetViewMode	Set view mode for scope
xPCScopes.TargetScopeSetYLimits	Set y-axis limits for scope

## Parameters

xPCTarget.GetNumParams	Get number of tunable parameters
xPCTarget.GetParam	Get parameter values
xPCTarget.GetParamDims	Get row and column dimensions of parameter
xPCTarget.GetParamIdx	Get parameter index
xPCTarget.GetParamName	Get parameter name
xPCTarget.SetParam	Change parameter value

## Signals

xPCTarget.GetNumSignals	Get number of signals
xPCTarget.GetSignal	Get signal value
xPCTarget.GetSignalidsfromLabel	Get signal IDs from signal label
xPCTarget.GetSignalIdx	Get signal index
xPCTarget.GetSignalLabel	Get signal label
xPCTarget.GetSignalName	Copy signal name to character array

xPCTarget.GetSignals	Get vector of signal values
xPCTarget.GetSignalWidth	Get width of signal

## Data Logs

xPCTarget.GetNumStates	Get number of states
xPCTarget.GetOutputLog	Copy output log data to array
xPCTarget.GetStateLog	Get state log
xPCTarget.GetTETLog	Get TET log
xPCTarget.GetTimeLog	Get time log

## File Systems

FSDir	Type definition for file system folder information structure
FSDiskInfo	Type definition for file system disk information structure
xPCFileSystem.CD	Change current folder on target computer to specified path
xPCFileSystem.CloseFile	Close file on target computer
xPCFileSystem.DirList	Return contents of target computer folder
xPCFileSystem.GetDiskInfo	Return disk information
xPCFileSystem.GetFileSize	Return size of file on target computer
xPCFileSystem.Init	Initialize file system object to communicate with target computer
xPCFileSystem.MKDIR	Create folder on target computer
xPCFileSystem.OpenFile	Open file on target computer
xPCFileSystem.PWD	Get current folder of target computer
xPCFileSystem.ReadFile	Read open file on target computer

xPCFileSystem.RemoveFile	Remove file from target computer
xPCFileSystem.RMDIR	Remove folder from target computer
xPCFileSystem.ScGetFileName	Get name of file for scope
xPCFileSystem.ScGetWriteMode	Get write mode of file for scope
xPCFileSystem.ScGetWriteSize	Get block write size of data chunks
xPCFileSystem.ScSetFileName	Specify file name to contain signal data
xPCFileSystem.ScSetWriteMode	Specify when file allocation table entry is updated
xPCFileSystem.ScSetWriteSize	Specify that memory buffer collect data in multiples of write size
xPCFileSystem.WriteFile	Write to file on target computer

## **Errors**

xPCProtocol.GetxPCErrorMsg	Return error string
xPCProtocol.isxPCError	Return error status
xPCScopes.GetxPCError	Get error string
xPCScopes.isxPCError	Get error status
xPCTarget.GetxPCError	Get error string
xPCTarget.isxPCError	Return error status

## **COM API Methods – Alphabetical List**

# FSDir

---

**Purpose** Type definition for file system folder information structure

**Syntax**

```
typedef struct {  
    BSTR Name;  
    BSTR Date;  
    BSTR Time;  
    long Bytes;  
    long isdir;  
} FSDir;
```

**Fields**

<i>Name</i>	This value contains the name of the file or folder.
<i>Date</i>	This value contains the date the file or folder was last modified.
<i>Time</i>	This value contains the time the file or folder was last modified.
<i>Bytes</i>	This value contains the size of the file in bytes. If the element is a folder, this value is 0.
<i>isdir</i>	This value indicates if the element is a file (0) or folder (1). If it is a folder, <i>Bytes</i> has a value of 0.

**Description** The FSDir structure contains information for a folder in the file system.

**See Also** API method `xPCFileSystem.DirList`

**Purpose** Type definition for file system disk information structure

**Syntax**

```
typedef struct {
    BSTR Label;
    BSTR DriveLetter;
    BSTR Reserved;
    long SerialNumber;
    long FirstPhysicalSector;
    long FATType;
    long FATCount;
    long MaxDirEntries;
    long BytesPerSector;
    long SectorsPerCluster;
    long TotalClusters;
    long BadClusters;
    long FreeClusters;
    long Files;
    long FileChains;
    long FreeChains;
    long LargestFreeChain;
} FSDiskInfo;
```

**Fields**

<i>Label</i>	This value contains the zero-terminated string that contains the volume label. The string is empty if the volume has no label.
<i>DriveLetter</i>	This value contains the drive letter, in uppercase.
<i>Reserved</i>	Reserved.
<i>SerialNumber</i>	This value contains the volume serial number.
<i>FirstPhysicalSector</i>	This value contains the logical block address (LBA) of the logical drive boot record. For 3.5-inch disks, this value is 0.

<i>FATType</i>	This value contains the type of file system found. It can contain 12 , 16 , or 32 for FAT-12, FAT-16, or FAT-32 volumes, respectively.
<i>FATCount</i>	This value contains the number of FAT partitions on the volume.
<i>MaxDirEntries</i>	This value contains the size of the root folder. For FAT-32 systems, this value is 0.
<i>BytesPerSector</i>	This value contains the sector size. This value is most likely to be 512.
<i>SectorsPerCluster</i>	This value contains, in sectors, the size of the smallest unit of storage that can be allocated to a file.
<i>TotalClusters</i>	This value contains the number of file storage clusters on the volume.
<i>BadClusters</i>	This value contains the number of clusters that have been marked as bad. These clusters are unavailable for file storage.
<i>FreeClusters</i>	This value contains the number of clusters that are currently available for storage.
<i>Files</i>	This value contains the number of files, including folders, on the volume. This number excludes the root folder and files that have an allocated file size of 0.
<i>FileChains</i>	This value contains the number of contiguous cluster chains. On a completely unfragmented volume, this value is identical to the value of <i>Files</i> .



<i>FreeChains</i>	This value contains the number of contiguous cluster chains of free clusters. On a completely unfragmented volume, this value is 1.
<i>LargestFreeChain</i>	This value contains the maximum allocated file size, in number of clusters, for a newly allocated contiguous file. On a completely unfragmented volume, this value is identical to <i>FreeClusters</i> .

**Description** The FSDiskInfo structure contains information for file system disks.

**See Also** API method `xPCFileSystem.GetDiskInfo`

# xPCFileSystem.CD

---

<b>Purpose</b>	Change current folder on target computer to specified path
<b>Prototype</b>	<code>long CD(BSTR <i>dir</i>);</code>
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem
<b>Arguments</b>	[in] <i>dir</i> Enter the path on the target computer to change to.
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.
<b>Description</b>	The xPCFileSystem.CD method changes the current folder on the target computer to the path specified in <i>dir</i> . Use the xPCFileSystem.PWD method to show the current folder of the target computer.
<b>See Also</b>	API method xPCFileSystem.PWD

<b>Purpose</b>	Close file on target computer		
<b>Prototype</b>	<code>CloseFile(long <i>filehandle</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem		
<b>Arguments</b>	<table><tr><td>[in] <i>filehandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr></table>	[in] <i>filehandle</i>	Enter the file handle of an open file on the target computer.
[in] <i>filehandle</i>	Enter the file handle of an open file on the target computer.		
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.		
<b>Description</b>	The <code>xPCFileSystem.CloseFile</code> method closes the file associated with <i>fileHandle</i> on the target computer. <i>fileHandle</i> is the handle of a file previously opened by the <code>xPCFileSystem.OpenFile</code> method.		
<b>See Also</b>	API methods <code>xPCFileSystem.OpenFile</code> , <code>xPCFileSystem.ReadFile</code> , <code>xPCFileSystem.WriteFile</code>		

# xPCFileSystem.DirList

---

**Purpose** Return contents of target computer folder

**Prototype** `DirList(BSTR path);`

**Member Of** XPCAPICOMLib.xPCFileSystem

**Arguments** [in] *path* Enter the path of the folder.

**Description** The `xPCFileSystem.DirList` method returns the contents of the target computer folder specified by *path* as an array of the `FSDir` structure.

**See Also** API structure `FSDir`  
API method `xPCFileSystem.GetDiskInfo`

<b>Purpose</b>	Return disk information		
<b>Prototype</b>	<code>GetDiskInfo(BSTR <i>driveLetter</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem		
<b>Arguments</b>	<table><tr><td>[in] <i>driveLetter</i></td><td>Enter the driver letter that contains the file system.</td></tr></table>	[in] <i>driveLetter</i>	Enter the driver letter that contains the file system.
[in] <i>driveLetter</i>	Enter the driver letter that contains the file system.		
<b>Description</b>	The <code>xPCFileSystem.GetDiskInfo</code> method accepts as input the drive specified by <i>driveLetter</i> and fills in the fields of the <code>FSDiskInfo</code> structure.		
<b>See Also</b>	API structure <code>FSDiskInfo</code> API method <code>xPCFileSystem.DirList</code>		

# xPCFileSystem.GetFileSize

---

<b>Purpose</b>	Return size of file on target computer		
<b>Prototype</b>	<code>long GetFileSize(long <i>filehandle</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem		
<b>Arguments</b>	<table><tr><td><code>[in] <i>filehandle</i></code></td><td>Enter the file handle of an open file on the target computer.</td></tr></table>	<code>[in] <i>filehandle</i></code>	Enter the file handle of an open file on the target computer.
<code>[in] <i>filehandle</i></code>	Enter the file handle of an open file on the target computer.		
<b>Return</b>	This method returns the size of the specified file in bytes.		
<b>Description</b>	The <code>xPCFileSystem.GetFileSize</code> method returns the size, in bytes, of the file associated with <i>filehandle</i> on the target computer. <i>filehandle</i> is the handle of a file previously opened by the <code>xPCFileSystem.OpenFile</code> method.		
<b>See Also</b>	API methods <code>xPCFileSystem.OpenFile</code> , <code>xPCFileSystem.ReadFile</code>		

<b>Purpose</b>	Initialize file system object to communicate with target computer		
<b>Prototype</b>	<code>long Init(IxPCProtocol* xPCProtocol);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem		
<b>Arguments</b>	<table><tr><td>[in] xPCProtocol</td><td>Specify the communication port of the target computer object for which the file system is to be initialized.</td></tr></table>	[in] xPCProtocol	Specify the communication port of the target computer object for which the file system is to be initialized.
[in] xPCProtocol	Specify the communication port of the target computer object for which the file system is to be initialized.		
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the xPCFileSystem.Init method returns 0.		
<b>Description</b>	The xPCFileSystem.Init method initializes the file system object to communicate with the target computer referenced by the xPCProtocol object.		

# xPCFileSystem.MKDIR

---

**Purpose** Create folder on target computer

**Prototype** long MKDIR(BSTR *dirname*);

**Member Of** XPCAPICOMLib.xPCFileSystem

**Arguments** [in] *dirname* Enter the name of the folder to create on the target computer.

**Return** If the method detects an error, it returns -1. Otherwise, the method returns 0.

**Description** The xPCFileSystem.MKDIR method creates the folder *dirname* in the current folder of the target computer.

**See Also** API method xPCFileSystem.PWD



**Purpose** Open file on target computer

**Prototype** long OpenFile(BSTR *filename*, BSTR *permission*);

**Member Of** XPCAPICOMLib.xPCFileSystem

**Arguments**

[in] <i>filename</i>	Enter the name of the file to open on the target computer.
[in] <i>permission</i>	Enter the read/write permission with which to open the file. Values are r (read) or w (read/write).

**Return** The xPCFileSystem.OpenFile method returns the file handle for the opened file.

**Description** The xPCFileSystem.OpenFile method opens the specified file, *filename*, on the target computer. If the file does not exist, the xPCFileSystem.OpenFile method creates *filename*, then opens it. You can open a file for read or read/write access.

---

**Note** Opening the file for write access overwrites the existing contents of the file. It does not append the new data.

---

**See Also** API methods xPCFileSystem.CloseFile, xPCFileSystem.GetFileSize, xPCFileSystem.ReadFile, xPCFileSystem.WriteFile

# xPCFileSystem.PWD

---

<b>Purpose</b>	Get current folder of target computer
<b>Prototype</b>	BSTR PWD();
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem
<b>Return</b>	This method returns the path of the current folder on the target computer.
<b>Description</b>	The xPCFileSystem.PWD method places the path of the current folder on the target computer.
<b>See Also</b>	API method xPCFileSystem.CD

<b>Purpose</b>	Read open file on target computer						
<b>Prototype</b>	VARIANT ReadFile(int <i>fileHandle</i> , int <i>start</i> , int <i>numbytes</i> );						
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem						
<b>Arguments</b>	<table><tr><td>[in] <i>fileHandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr><tr><td>[in] <i>start</i></td><td>Enter an offset from the beginning of the file from which this method can start to read.</td></tr><tr><td>[in] <i>numbytes</i></td><td>Enter the number of bytes this method is to read from the file.</td></tr></table>	[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.	[in] <i>start</i>	Enter an offset from the beginning of the file from which this method can start to read.	[in] <i>numbytes</i>	Enter the number of bytes this method is to read from the file.
[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.						
[in] <i>start</i>	Enter an offset from the beginning of the file from which this method can start to read.						
[in] <i>numbytes</i>	Enter the number of bytes this method is to read from the file.						
<b>Return</b>	This method returns the results of the read operation as a VARIANT of type Byte. If the method detects an error, it returns VT_ERROR, whose value is 10, instead.						
<b>Description</b>	The xPCFileSystem.ReadFile method reads an open file on the target computer and returns the results of the read operation as a VARIANT of type Byte. <i>fileHandle</i> is the file handle of a file previously opened by xPCFileSystem.OpenFile. You can specify that the read operation begin at the beginning of the file (default) or at a certain offset into the file ( <i>start</i> ). The <i>numbytes</i> parameter specifies how many bytes the xPCFileSystem.ReadFile method is to read from the file.						
<b>See Also</b>	API methods xPCFileSystem.CloseFile, xPCFileSystem.GetFileSize, xPCFileSystem.OpenFile, xPCFileSystem.WriteFile						

# xPCFileSystem.RemoveFile

---

<b>Purpose</b>	Remove file from target computer		
<b>Prototype</b>	<code>long RemoveFile(BSTR <i>filename</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem		
<b>Arguments</b>	<table><tr><td><code>[in] <i>filename</i></code></td><td>Enter the name of a file on the target computer.</td></tr></table>	<code>[in] <i>filename</i></code>	Enter the name of a file on the target computer.
<code>[in] <i>filename</i></code>	Enter the name of a file on the target computer.		
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.		
<b>Description</b>	The <code>xPCFileSystem.RemoveFile</code> method removes the file named <i>filename</i> from the target computer file system. <i>filename</i> can be a relative or absolute path name on the target computer.		

<b>Purpose</b>	Remove folder from target computer		
<b>Prototype</b>	<code>long RMDIR(BSTR <i>dirname</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem		
<b>Arguments</b>	<table><tr><td><code>[in] <i>dirname</i></code></td><td>Enter the name of a folder on the target computer.</td></tr></table>	<code>[in] <i>dirname</i></code>	Enter the name of a folder on the target computer.
<code>[in] <i>dirname</i></code>	Enter the name of a folder on the target computer.		
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.		
<b>Description</b>	The xPCFileSystem.RMDIR method removes a folder named <i>dirname</i> from the target computer file system. <i>dirname</i> can be a relative or absolute path name on the target computer.		

# xPCFileSystem.ScGetFileName

---

<b>Purpose</b>	Get name of file for scope
<b>Prototype</b>	BSTR ScGetFileName(long <i>scNum</i> );
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem
<b>Arguments</b>	[in] <i>scNum</i> Enter the scope number.
<b>Return</b>	Returns the name of the file for the scope.
<b>Description</b>	The xPCFileSystem.ScGetFileName method returns the name of the file to which scope <i>scNum</i> will save signal data.
<b>See Also</b>	API method xPCFileSystem.ScSetFileName

<b>Purpose</b>	Get write mode of file for scope
<b>Prototype</b>	<code>long ScGetWriteMode(long <i>scNum</i>);</code>
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem
<b>Arguments</b>	[in] <i>scNum</i> Enter the scope number.
<b>Return</b>	This method returns the number indicating the write mode. Values are  0      Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).  1      Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.
<b>Description</b>	The xPCFileSystem.ScGetWriteMode method returns the write mode of the file for the scope.
<b>See Also</b>	API method xPCFileSystem.ScSetWriteMode

# xPCFileSystem.ScGetWriteSize

---

**Purpose** Get block write size of data chunks

**Prototype** `long ScGetWriteSize(long scNum);`

**Member Of** XPCAPICOMLib.xPCFileSystem

**Arguments** [in] *scNum* Enter the scope number.

**Return** This method returns the block size, in bytes, of the data chunks.

**Description** The xPCFileSystem.ScGetWriteSize method gets the block size, in bytes, of the data chunks.

**See Also** API method xPCFileSystem.ScSetWriteSize



<b>Purpose</b>	Specify file name to contain signal data				
<b>Prototype</b>	<code>long ScSetFileName(long <i>scNum</i>, BSTR <i>filename</i>);</code>				
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>filename</i></td><td>Enter the name of a file to contain the signal data.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>filename</i>	Enter the name of a file to contain the signal data.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>filename</i>	Enter the name of a file to contain the signal data.				
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.				
<b>Description</b>	The <code>xPCFileSystem.ScSetFileName</code> method sets the name of the file to which the scope will save the signal data. The xPC Target software creates this file in the target computer file system. Note that you can only call this method when the scope is stopped.				
<b>See Also</b>	API method <code>xPCFileSystem.ScGetFileName</code>				

# xPCFileSystem.ScSetWriteMode

---

**Purpose** Specify when file allocation table entry is updated

**Prototype** `long ScSetWriteMode(long scNum, long writeMode);`

**Member Of** XPCAPICOMLib.xPCFileSystem

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>writeMode</i>	Enter an integer for the write mode:
0	Enables lazy write mode
1	Enables commit write mode

**Return** If the method detects an error, it returns -1. Otherwise, the method returns 0.

**Description** The `xPCFileSystem.ScSetWriteMode` method specifies when a file allocation table (FAT) entry is updated. Both modes write the signal data to the file, as follows:

- 0 Lazy mode. The FAT entry is updated only when the file is closed and not during each file write operation. This mode is faster, but if the system crashes before the file is closed, the file system might not have the actual file size (the file contents, however, will be intact).
- 1 Commit mode. Each file write operation simultaneously updates the FAT entry for the file. This mode is slower, but the file system maintains the actual file size.

**See Also** API method `xPCFileSystem.ScSetWriteMode`  
Scope object property `Mode`

<b>Purpose</b>	Specify that memory buffer collect data in multiples of write size				
<b>Prototype</b>	<code>long ScSetWriteSize(long <i>scNum</i>, long <i>writeSize</i>);</code>				
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>writeSize</i></td><td>Enter the block size, in bytes, of the data chunks.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>writeSize</i>	Enter the block size, in bytes, of the data chunks.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>writeSize</i>	Enter the block size, in bytes, of the data chunks.				
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.				
<b>Description</b>	The <code>xPCFileSystem.ScSetWriteSize</code> method specifies that a memory buffer collect data in multiples of <i>writeSize</i> . By default, this parameter is 512 bytes, which is the typical disk sector size. Using a block size that is the same as the disk sector size provides better performance. <i>writeSize</i> must be a multiple of 512.				
<b>See Also</b>	API method <code>xPCFileSystem.ScGetWriteSize</code> Scope object property <code>WriteSize</code>				

# xPCFileSystem.WriteFile

---

<b>Purpose</b>	Write to file on target computer						
<b>Prototype</b>	<code>long WriteFile(long <i>fileHandle</i>, long <i>numbytes</i>, VARIANT <i>buffer</i>);</code>						
<b>Member Of</b>	XPCAPICOMLib.xPCFileSystem						
<b>Arguments</b>	<table><tr><td>[in] <i>fileHandle</i></td><td>Enter the file handle of an open file on the target computer.</td></tr><tr><td>[in] <i>numbytes</i></td><td>Enter the number of bytes this method is to write into the file.</td></tr><tr><td>[in] <i>buffer</i></td><td>The contents to write to <i>fileHandle</i> are stored in <i>buffer</i>.</td></tr></table>	[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.	[in] <i>numbytes</i>	Enter the number of bytes this method is to write into the file.	[in] <i>buffer</i>	The contents to write to <i>fileHandle</i> are stored in <i>buffer</i> .
[in] <i>fileHandle</i>	Enter the file handle of an open file on the target computer.						
[in] <i>numbytes</i>	Enter the number of bytes this method is to write into the file.						
[in] <i>buffer</i>	The contents to write to <i>fileHandle</i> are stored in <i>buffer</i> .						
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, the method returns 0.						
<b>Description</b>	The xPCFileSystem.WriteFile method writes the contents of the VARIANT <i>buffer</i> , of type Byte, to the file specified by <i>fileHandle</i> on the target computer. The <i>fileHandle</i> parameter is the handle of a file previously opened by xPCFSOpenFile. <i>numbytes</i> is the number of bytes to write to the file.						
<b>See Also</b>	API methods xPCFileSystem.CloseFile, xPCFileSystem.GetFileSize, xPCFileSystem.OpenFile, xPCFileSystem.ReadFile						

<b>Purpose</b>	Close RS-232 or TCP/IP communication connection
<b>Prototype</b>	long Close();
<b>Member Of</b>	XPCAPICOMLib.xPCProtocol
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	The xPCProtocol.Close method closes the communication channel opened by xPCProtocol.RS232Connect or xPCProtocol.TcpIpConnect.

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

# xPCProtocol.GetLoadTimeOut

---

<b>Purpose</b>	Return current timeout value for target application initialization
<b>Prototype</b>	<code>long GetLoadTimeOut();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCProtocol</code>
<b>Return</b>	If the method detects an error, it returns -1. Otherwise, it returns the number of seconds allowed for the initialization of the target application.
<b>Description</b>	<p>The <code>xPCProtocol.GetLoadTimeOut</code> method returns the number of seconds allowed for the initialization of the target application.</p> <p>When you load a new target application onto the target computer, the method <code>xPCTarget.LoadApp</code> waits for a certain amount of time before checking to see whether the initialization of the target application is complete. In the case where initialization of the target application is not complete, the method <code>xPCTarget.LoadApp</code> returns a timeout error. By default, <code>xPCTarget.LoadApp</code> checks five times to see whether the target application is ready, with each attempt taking about 1 second. However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout is generated. The method <code>xPCProtocol.SetLoadTimeOut</code>/<code>xPCProtocol.SetLoadTimeOut</code> sets the timeout to a different number.</p> <p>Use the <code>xPCProtocol.GetLoadTimeOut</code> method if you suspect that the current number of seconds (the timeout value) is too short. Then use the <code>xPCProtocol.SetLoadTimeOut</code> method to set the timeout to a higher number.</p>

<b>Purpose</b>	Return error string
<b>Prototype</b>	BSTR GetxPCErrorMsg();
<b>Member Of</b>	XPCAPICOMLib.xPCProtocol
<b>Return</b>	If the xPCProtocol.GetxPCErrorMsg method completes without detecting an error, it returns the string for the last reported error.
<b>Description</b>	The xPCProtocol.GetxPCErrorMsg method returns the string of the last error reported by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the xPCProtocol.isxPCError method, which detects that an error has occurred.
<b>See Also</b>	API function xPCProtocol.isxPCError

# xPCProtocol.Init

---

<b>Purpose</b>	Initialize xPC Target API DLL
<b>Prototype</b>	<code>long Init();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCProtocol</code>
<b>Return</b>	If the xPC Target DLL, <code>xpcapi.dll</code> loads without causing <code>xPCProtocol.Init</code> to detect an error, the method returns 0. If <code>xpcapi.dll</code> fails to load, this method returns -1.
<b>Description</b>	<p>The <code>xPCProtocol.Init</code> method initializes the xPC Target API by loading the xPC Target DLL, <code>xpcapi.dll</code>, into memory. To load <code>xpcapi.dll</code> into memory, the method requires that the <code>xpcapi.dll</code> file be in one of the following folders:</p> <ul style="list-style-type: none"><li>• The folder in which the application is loaded</li><li>• The current folder</li><li>• The Windows system folder</li></ul>



<b>Purpose</b>	Return error status
<b>Prototype</b>	<code>long isxPCError();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCProtocol
<b>Return</b>	If an error occurred, the method returns 1. Otherwise, it returns 0.
<b>Description</b>	Use the <code>xPCProtocol.isxPCError</code> method to check for errors that might occur after a call to the <code>xPCProtocol</code> class methods. If the method detects that an error occurred, call the <code>xPCProtocol.GetxPCErrorMsg</code> to get the string for the error.
<b>See Also</b>	API function <code>xPCProtocol.GetxPCErrorMsg</code>

# xPCProtocol.Port

---

<b>Purpose</b>	Contain communication channel index
<b>Prototype</b>	<code>long Port();</code>
<b>Member Of</b>	<code>XPCAPICOMLIB.xPCProtocol</code>
<b>Return</b>	If the method detects an error, it returns a nonpositive number. Otherwise, it returns a positive number (the communication channel index).
<b>Description</b>	The <code>xPCProtocol.Port</code> property contains the communication channel index if connection with the target computer succeeds. Note that you only need to use this property when working with a model-specific COM library that you generate from a Simulink model. See “Model-Specific COM Interface Library (model_nameCOMiface.dll)”.

<b>Purpose</b>	Reboot target computer
<b>Prototype</b>	long Reboot();
<b>Member Of</b>	XPCAPICOMLib.xPCProtocol
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	The xPCProtocol.Reboot method reboots the target computer. This function does not close the connection to the target computer. You should explicitly close the connection, then reestablish the connection once the target computer has rebooted. Use the methods xPCProtocol.RS232Connect or xPCProtocol.TcpIpConnect to reestablish the connection.

# xPCProtocol.RS232Connect

---

**Purpose** Open RS-232 connection to target computer

**Prototype** `long RS232Connect(long comport, long baudrate);`

**Member Of** XPCAPICOMLib.xPCProtocol

**Arguments**

[in] <i>comport</i>	Index of the COM port to be used (0 is COM1, 1 is COM2, and so forth).
[in] <i>baudrate</i>	<i>baudrate</i> must be one of the following values: 1200, 2400, 4800, 9600, 19200, 38400, 57600, or 115200.

**Return** The `xPCProtocol.RS232Connect` method returns the port value for the connection. If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCProtocol.RS232Connect` method initiates an RS-232 connection to an xPC Target system. It returns the port value for the connection. Be sure to pass this value to every xPC Target API function that requires a port value.

If you enter a value of 0 for *baudrate*, this function sets the baud rate to the default value (115200).

---

**Note** RS-232 Host-Target communication mode will be removed in a future release. Use TCP/IP instead.

---

**Purpose** Change initialization timeout value

**Prototype** `long SetLoadTimeOut(long timeOut);`

**Member Of** XPCAPICOMLib.xPCProtocol

**Arguments** [in] *timeOut* Enter the new initialization timeout value.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1. To get the string description for the error, use `xPCProtocol.GetxPCErrorMsg`.

**Description** The `xPCProtocol.SetLoadTimeOut` method changes the timeout value for initialization. The *timeOut* value is the time the method `xPCTarget.LoadApp` waits to check whether the model initialization for a new application is complete before returning. It enables you to set the number of initialization attempts to be made before signaling a timeout. When a new target application is loaded onto the target computer, the method `xPCTarget.LoadApp` waits for a certain time to check whether the model initialization is complete before returning. If the model initialization is incomplete within the allotted time, `xPCTarget.LoadApp` returns a timeout error.

By default, `xPCTarget.LoadApp` checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can be generated.

# xPCProtocol.TargetPing

---

<b>Purpose</b>	Ping target computer
<b>Prototype</b>	long TargetPing;
<b>Member Of</b>	XPCAPICOMLIB.xPCProtocol
<b>Return</b>	The xPCProtocol.TargetPing method does not return an error status. This method returns 1 if it reaches the target computer and the computer responds. If the target computer does not respond, the method returns 0.
<b>Description</b>	<p>The xPCProtocol.TargetPing method pings the target computer and returns 1 or 0 depending on whether the target responds or not. Errors such as the inability to connect to the target are ignored.</p> <p>If you are using TCP/IP, note that xPCProtocol.TargetPing will cause the target computer to close the TCP/IP connection. You can use xPCProtocol.TcpIpConnect to reconnect. You can also use this xPCProtocol.TargetPing feature to close the target computer connection in the event of an aborted TCP/IP connection (for example, if your host side program crashes).</p>

<b>Purpose</b>	Open TCP/IP connection to target computer	
<b>Prototype</b>	<code>long TcpIpConnect(BSTR <i>TargetIpAddress</i>, BSTR <i>TargetPort</i>);</code>	
<b>Member Of</b>	XPCAPICOMLIB.xPCProtocol	
<b>Arguments</b>	[in] <i>TargetIpAddress</i>	Enter the IP address of the target as a dotted decimal string. For example, "192.168.0.10".
	[in] <i>TargetPort</i>	Enter the associated IP port as a string. For example, "22222".
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.	
<b>Description</b>	The <code>xPCProtocol.TcpIpConnect</code> method opens a connection to the TCP/IP location specified by the IP address. Use this integer as the <i>TargetPort</i> variable in the xPC Target COM API functions that require a port value.	

## xPCProtocol.Term

---

<b>Purpose</b>	Unload xPC Target API DLL from memory
<b>Prototype</b>	<code>long Term();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCProtocol</code>
<b>Return</b>	The <code>xPCProtocol.Term</code> method always returns -1.
<b>Description</b>	The <code>xPCProtocol.Term</code> method unloads the xPC Target API DLL ( <code>xpcapi.dll</code> ) from memory. You must call this method when you want to terminate your COM API application.



<b>Purpose</b>	Create new file scope		
<b>Prototype</b>	<code>long AddFileScope(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter a number for a new scope. Values are 1, 2, 3. . .</td></tr></table>	[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .
[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .		
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.		
<b>Description</b>	<p>The <code>xPCScopes.AddFileScope</code> method creates a new file scope on the target computer.</p> <p>Calling the <code>xPCScopes.AddFileScope</code> method with <i>scNum</i> having the number of an existing scope produces an error. Use <code>xPCScopes.GetScopes</code> to find the numbers of existing scopes.</p>		

# xPCScopes.AddHostScope

---

<b>Purpose</b>	Create new host scope		
<b>Prototype</b>	<code>long AddHostScope(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter a number for a new scope. Values are 1, 2, 3. . .</td></tr></table>	[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .
[in] <i>scNum</i>	Enter a number for a new scope. Values are 1, 2, 3. . .		
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.		
<b>Description</b>	<p>The <code>xPCScopes.AddHostScope</code> method creates a new host scope on the target computer.</p> <p>Calling the <code>xPCScopes.AddHostScope</code> method with <i>scNum</i> having the number of an existing scope produces an error. Use <code>xPCScopes.GetScopes</code> to find the numbers of existing scopes.</p>		

<b>Purpose</b>	Create new target scope		
<b>Prototype</b>	<code>long AddTargetScope(long <i>scNum</i>);</code>		
<b>Member Of</b>	<code>XPCAPICOMLib.xPCScopes</code>		
<b>Arguments</b>	<table><tr><td><code>[in] <i>scNum</i></code></td><td>Enter a number for a new scope. Values are 1, 2, 3. . .</td></tr></table>	<code>[in] <i>scNum</i></code>	Enter a number for a new scope. Values are 1, 2, 3. . .
<code>[in] <i>scNum</i></code>	Enter a number for a new scope. Values are 1, 2, 3. . .		
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.		
<b>Description</b>	<p>If the method detects an error, it returns 0. The <code>xPCScopes.AddTargetScope</code> method creates a new scope on the target computer.</p> <p>Calling the <code>xPCScopes.AddTargetScope</code> method with <i>scNum</i> having the number of an existing scope produces an error. Use <code>xPCScopes.GetScopes</code> to find the numbers of existing scopes.</p>		

# xPCScopes.GetScopes

---

**Purpose** Get and copy list of scope numbers

**Prototype** VARIANT GetScopes(long *size*);

**Member Of** XPCAPICOMLib.xPCScopes

**Arguments** [in] *size* Specify the size of the VARIANT array returned. This argument must be greater than MAX\_SCOPES-1. The elements in the array consist of a list of unsorted integers, terminated by -1.

**Return** The xPCScopes.GetScopes method returns a VARIANT array with elements containing a list of scope numbers from the target application.

**Description** The xPCScopes.GetScopes method gets a VARIANT array with elements containing a list of scope numbers currently defined for the target application. Specify the size of the VARIANT array returned. This size must be greater than the maximum number of scopes - 1, up to a maximum of 30 scopes. The elements in the array consist of a list of unsorted integers, terminated by -1.

<b>Purpose</b>	Get error string
<b>Prototype</b>	BSTR GetxPCError();
<b>Member Of</b>	XPCAPICOMLib.xPCScopes
<b>Return</b>	The xPCScopes.GetxPCError method returns the string for the last reported error. If the software has not reported an error, this method returns 0.
<b>Description</b>	The xPCScopes.GetxPCError method gets the string of the last reported error by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the xPCScopes.isxPCError method, which detects that an error has occurred.
<b>See Also</b>	API function xPCScopes.isxPCError

# xPCScopes.Init

---

<b>Purpose</b>	Initialize scope object to communicate with target computer		
<b>Prototype</b>	<code>long Init(IxPCProtocol* xPCProtocol);</code>		
<b>Member Of</b>	XPCAPICOMLib.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] xPCProtocol</td><td>Specify the communication port of the target computer object for which the scope is to be initialized.</td></tr></table>	[in] xPCProtocol	Specify the communication port of the target computer object for which the scope is to be initialized.
[in] xPCProtocol	Specify the communication port of the target computer object for which the scope is to be initialized.		
<b>Return</b>	If the xPCScopes.Init method initializes the scope object without detecting an error, it returns 0. If the scope object fails to initialize, the method returns -1.		
<b>Description</b>	The xPCScopes.Init method initializes the scope object to communicate with the target computer referenced by the xPCProtocol object.		

<b>Purpose</b>	Get data acquisition status for scope		
<b>Prototype</b>	<code>long IsScopeFinished(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
<b>Return</b>	If the method detects an error, it returns -1. If a scope finishes a data acquisition cycle, this method returns 1. If the scope is in the process of acquiring data, this method returns 0.		
<b>Description</b>	The <code>xPCScopes.IsScopeFinished</code> method gets a 1 or 0 depending on whether scope <i>scNum</i> is finished (state of <code>SCST_FINISHED</code> ) or not. You can also call this function for target scopes; however, because target scopes restart immediately, it is almost impossible to find these scopes in the finished state.		

# xPCScopes.isxPCError

---

<b>Purpose</b>	Get error status
<b>Prototype</b>	<code>long isxPCError();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes
<b>Return</b>	If an error occurred, the method returns 1. Otherwise, it returns 0.
<b>Description</b>	Use the <code>xPCScopes.isxPCError</code> method to check for errors that might occur after a call to the <code>xPCScopes</code> class methods. If the software detects that an error occurred, call the <code>xPCScopes.GetxPCError</code> method to get the string for the error.
<b>See Also</b>	API function <code>xPCScopes.GetxPCError</code>



<b>Purpose</b>	Remove scope		
<b>Prototype</b>	<code>long RemScope(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.		
<b>Description</b>	The <code>xPCScopes.RemScope</code> method removes the scope with number <i>scNum</i> . Attempting to remove a nonexistent scope causes an error. For a list of existing scopes, use <code>xPCScopes.GetScopes</code> .		

# xPCScopes.ScopeAddSignal

---

**Purpose** Add signal to scope

**Prototype** `long ScopeAddSignal(long scNum, long sigNum);`

**Member Of** XPCAPICOMLib.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>sigNum</i>	Enter a signal number.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeAddSignal` method adds the signal with number *sigNum* to the scope *scNum*. The signal should not already exist in the scope. You can use `xPCScopes.ScopeGetSignals` to get a list of the signals already present. Use the `xPCTarget.GetSignalIdx` method to get the signal number.

<b>Purpose</b>	Scope autorestart value		
<b>Prototype</b>	<code>long ScopeGetAutoRestart(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
<b>Return</b>	The <code>xPCScopes.ScopeGetAutoRestart</code> method returns the scope autorestart flag value (1 if enabled, 0 if disabled). If the method detects an error, it returns -1.		
<b>Description</b>	The <code>xPCScopes.ScopeGetAutoRestart</code> method gets the autorestart flag value for scope <i>scNum</i> . Autorestart flag can be disabled (0) or enabled (1).		

# xPCScopes.ScopeGetData

---

**Purpose** Copy scope data to array

**Prototype** VARIANT ScopeGetData(long *scNum*, long *signal\_id*, long *start*, long *numsamples*, long *decimation*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>signal_id</i>	Enter a signal number. Enter -1 to get time stamped data.
[in] <i>start</i>	Enter the first sample from which data retrieval is to start.
[in] <i>numsamples</i>	Enter the number of samples retrieved with a decimation of <i>decimation</i> , starting from the <i>start</i> value.
[in] <i>decimation</i>	Enter a value such that every <i>decimation</i> sample is retrieved in a scope window.

**Return** The xPCScopes.ScopeGetData method returns a VARIANT array with elements containing the data used in a scope.

**Description** The xPCScopes.ScopeGetData method gets the data used in a scope. Use this function for scopes of type SCTYPE\_HOST. The scope must be either in state Finished or in state Interrupted for the data to be retrievable. (Use the xPCScopes.ScopeGetState method to check the state of the scope.) The data must be retrieved one signal at a time. The calling function determines and allocates the space ahead of time to store the scope data. Use the method xPCScopes.ScopeGetSignals to get the list of signals in the scope for *signal\_id*.

To get time stamped data, specify -1 for `signal_id`. From the output, you can then get the number of nonzero elements.

# xPCScopes.ScopeGetDecimation

---

**Purpose** Get decimation of scope

**Prototype** `long ScopeGetDecimation(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The xPCScopes.ScopeGetDecimation method returns the decimation of scope *scNum*. If the method detects an error, it returns -1.

**Description** The xPCScopes.ScopeGetDecimation method gets the decimation of scope *scNum*. The decimation is a number, N, meaning every Nth sample is acquired in a scope window.

# xPCScopes.ScopeGetNumPrePostSamples

---

<b>Purpose</b>	Get number of pre- or posttriggering samples before triggering scope		
<b>Prototype</b>	<code>long ScopeGetNumPrePostSamples(long scNum);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
<b>Return</b>	The <code>xPCScopes.ScopeGetNumPrePostSamples</code> method returns the number of samples for pre- or posttriggering for scope <i>scNum</i> . If an error occurs, this method returns -1.		
<b>Description</b>	The <code>xPCScopes.ScopeGetNumPrePostSamples</code> method gets the number of samples for pre- or posttriggering for scope <i>scNum</i> . A negative number implies pretriggering, whereas a positive number implies posttriggering samples.		

# xPCScopes.ScopeGetNumSamples

---

**Purpose** Get number of samples in one data acquisition cycle

**Prototype** `long ScopeGetNumSamples(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetNumSamples` method returns the number of samples in the scope *scNum*. If the method detects an error, it returns -1.

**Description** The `xPCScopes.ScopeGetNumSamples` method gets the number of samples in one data acquisition cycle for scope *scNum*.



**Purpose** Get list of signals

**Prototype** VARIANT ScopeGetSignals(long *scNum*, long *size*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>size</i>	Enter an integer to allocate the number of elements to be returned in the VARIANT array. This size is required for the method to copy the list of signals into the VARIANT array. The maximum number of signals is 10.

**Return** The xPCScopes.ScopeGetSignals method returns a VARIANT array with elements consisting of the list of signals defined for a scope.

**Description** The xPCScopes.ScopeGetSignals method gets the list of signals defined for scope *scNum*. You can use the constant MAX\_SIGNALS.

# xPCScopes.ScopeGetStartTime

---

**Purpose** Get last data acquisition cycle start time

**Prototype** `double ScopeGetStartTime(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetStartTime` method returns the start time for the last data acquisition cycle of a scope. If the method detects an error, it returns -1.

**Description** The `xPCScopes.ScopeGetStartTime` method gets the time at which the last data acquisition cycle for scope *scNum* started. This is only valid for scopes of type `SCTYPE_HOST`.

**Purpose** Get state of scope

**Prototype** BSTR ScopeGetState(long *scNum*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The xPCScopes.ScopeGetState method returns the state of scope *scNum*. If the method detects an error, it returns -1.

**Description** The xPCScopes.ScopeGetState method gets the state of scope *scNum*, or -1 upon error.

Constants to find the scope state have the following meanings:

Constant	Value	Description
SCST_WAITTOSTART	0	Scope is ready and waiting to start.
SCST_PREACQUIRING	5	Scope acquires a predefined number of samples before triggering.
SCST_WAITFORTRIG	1	After a scope is finished with the preacquiring state, it waits for a trigger. If the scope does not preacquire data, it enters the wait for trigger state.
SCST_ACQUIRING	2	Scope is acquiring data. The scope enters this state when it leaves the wait for trigger state.

## xPCScopes.ScopeGetState

---

Constant	Value	Description
SCST_FINISHED	3	Scope is finished acquiring data when it has attained the predefined limit.
SCST_INTERRUPTED	4	The user has stopped (interrupted) the scope.

# xPCScopes.ScopeGetTriggerLevel

---

**Purpose** Get trigger level for scope

**Prototype** `double ScopeGetTriggerLevel(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetTriggerLevel` method returns the scope trigger level. If the method detects an error, it returns -1.

**Description** The `xPCScopes.ScopeGetTriggerLevel` method gets the trigger level for scope *scNum*.

# xPCScopes.ScopeGetTriggerMode

---

**Purpose** Get trigger mode for scope

**Prototype** `long ScopeGetTriggerMode(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetTriggerMode` method returns the scope trigger mode. If the method detects an error, it returns -1.

**Description** The `xPCScopes.ScopeGetTriggerMode` method gets the trigger mode for scope *scNum*. Use the constants here to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

**See Also**

API function `xPCScopes.ScopeGetTriggerModeStr`

# xPCScopes.ScopeGetTriggerModeStr

---

**Purpose** Get trigger mode as string

**Prototype** `BSTR ScopeGetTriggerModeStr(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetTriggerModeStr` method returns a string containing the trigger mode string.

**Description** The `xPCScopes.ScopeGetTriggerModeStr` method gets the trigger mode string for scope *scNum*. This method returns one of the following strings.

Constant	Description
FreeRun	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances.
Software	Only user intervention can trigger the scope. No other triggering is possible.
Signal	The scope is triggered only after a signal has crossed a value.
Scope	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

**See Also** API function `xPCScopes.ScopeGetTriggerMode`



# xPCScopes.ScopeGetTriggerSample

---

<b>Purpose</b>	Get sample number for triggering scope		
<b>Prototype</b>	<code>long ScopeGetTriggerSample(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
<b>Return</b>	The <code>xPCScopes.ScopeGetTriggerSample</code> method returns a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope. If the method detects an error, it returns -1.		
<b>Description</b>	The <code>xPCScopes.ScopeGetTriggerSample</code> method gets the number of samples a triggering scope ( <i>scNum</i> ) acquires before starting data acquisition on a second scope. This value is a nonnegative integer for a real sample, and -1 for the special case where triggering is at the end of the data acquisition cycle for a triggering scope.		

# xPCScopes.ScopeGetTriggerSignal

---

**Purpose** Get trigger signal for scope

**Prototype** `long ScopeGetTriggerSignal(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The xPCScopes.ScopeGetTriggerSignal method returns the scope trigger signal. If the method detects an error, it returns -1.

**Description** The xPCScopes.ScopeGetTriggerSignal method gets the trigger signal for scope *scNum*.

**Purpose** Get trigger slope for scope

**Prototype** `long ScopeGetTriggerSlope(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetTriggerSlope` method returns the scope trigger slope. If the method detects an error, it returns -1.

**Description** The `xPCScopes.ScopeGetTriggerSlope` method gets the trigger slope of scope *scNum*. Use the constants here to interpret the trigger slope:

String	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger slope must be rising when the signal crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger slope must be falling when the signal crosses the trigger value.

**See Also** API function `xPCScopes.ScopeGetTriggerSlopeStr`

# xPCScopes.ScopeGetTriggerSlopeStr

---

**Purpose** Get trigger slope as string

**Prototype** `BSTR ScopeGetTriggerSlopeStr(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.ScopeGetTriggerSlopeStr` method returns a string containing the trigger slope string.

**Description** The `xPCScopes.ScopeGetTriggerSlopeStr` method gets the trigger slope string for scope *scNum*. This method returns one of the following strings:

String	Description
Either	The trigger slope can be either rising or falling.
Rising	The trigger slope must be rising when the signal crosses the trigger value.
Falling	The trigger slope must be falling when the signal crosses the trigger value.

**See Also** API function `xPCScopes.ScopeGetTriggerSlope`

**Purpose** Get type of scope

**Prototype** BSTR ScopeGetType(long *scNum*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The xPCScopes.ScopeGetType method returns the scope type as a string. If the method detects an error, it returns -1.

**Description** The xPCScopes.ScopeGetType method gets the type of scope *scNum*. This method returns one of the following strings:

String	Description
HOST	Host scope
Target	Target scope

# xPCScopes.ScopeRemSignal

---

**Purpose** Remove signal from scope

**Prototype** `long ScopeRemSignal(long scNum, long sigNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
-------------------	-------------------------

[in] <i>sigNum</i>	Enter a signal number.
--------------------	------------------------

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeRemSignal` method removes a signal from the scope with number *scNum*. The scope must already exist, and signal number *sigNum* must exist in the scope. Use `xPCScopes.GetScopes` to determine the existing scopes, and use `xPCScopes.ScopeGetSignals` to determine the existing signals for a scope. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

<b>Purpose</b>	Scope autorestart value				
<b>Prototype</b>	<code>long ScopeSetAutoRestart(long <i>scNum</i>, long <i>onoff</i>);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>onoff</i></td><td>Enter value to enable (1) or disable (0) scope autorestart.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>onoff</i>	Enter value to enable (1) or disable (0) scope autorestart.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>onoff</i>	Enter value to enable (1) or disable (0) scope autorestart.				
<b>Return</b>	The <code>xPCScopes.ScopeSetAutoRestart</code> method returns the scope autorestart flag value (1 if enabled, 0 if disabled). If the method detects an error, it returns -1.				
<b>Description</b>	The <code>xPCScopes.ScopeSetAutoRestart</code> method sets the autorestart flag value for scope <i>scNum</i> . Autorestart flag can be disabled (0) or enabled (1).				

# xPCScopes.ScopeSetDecimation

---

<b>Purpose</b>	Set decimation of scope				
<b>Prototype</b>	<code>long ScopeSetDecimation(long <i>scNum</i>, long <i>decimation</i>);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>decimation</i></td><td>Enter an integer for the decimation.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>decimation</i>	Enter an integer for the decimation.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>decimation</i>	Enter an integer for the decimation.				
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.				
<b>Description</b>	The <code>xPCScopes.ScopeSetDecimation</code> method sets the <i>decimation</i> of scope <i>scNum</i> . The decimation is a number, N, meaning every Nth sample is acquired in a scope window. Use this function only when the scope is stopped. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope.				



# xPCScopes.ScopeSetNumPrePostSamples

---

**Purpose** Set number of pre- or posttriggering samples before triggering scope

**Prototype** `long ScopeSetNumPrePostSamples(long scNum, long prepost);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>prepost</i>	A negative number means pretriggering, while a positive number means posttriggering. This function can only be used when the scope is stopped.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeSetNumPrePostSamples` method sets the number of samples for pre- or posttriggering for scope *scNum* to *prepost*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scope numbers.

# xPCScopes.ScopeSetNumSamples

---

**Purpose** Set number of samples in one data acquisition cycle

**Prototype** `long ScopeSetNumSamples(long scNum, long samples);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>samples</i>	Enter the number of samples you want to acquire in one cycle.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeSetNumSamples` method sets the number of samples for scope *scNum* to *samples*. Use this function only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope.

# xPCScopes.ScopeSetTriggerLevel

---

<b>Purpose</b>	Set trigger level for scope				
<b>Prototype</b>	<code>long ScopeSetTriggerLevel(long <i>scNum</i>, double <i>level</i>);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>level</i></td><td>Value for a signal to trigger data acquisition with a scope.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>level</i>	Value for a signal to trigger data acquisition with a scope.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>level</i>	Value for a signal to trigger data acquisition with a scope.				
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.				
<b>Description</b>	The <code>xPCScopes.ScopeSetTriggerLevel</code> method sets the trigger level to <i>level</i> for scope <i>scNum</i> . Use this function only when the scope is stopped. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope.				

# xPCScopes.ScopeSetTriggerMode

---

**Purpose** Set trigger mode of scope

**Prototype** `long ScopeSetTriggerMode(long scNum, long triggermode);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>triggermode</i>	Trigger mode for a scope.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeSetTriggerMode` method sets the trigger mode of scope *scNum* to *triggermode*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes. Use the constants defined here to interpret the trigger mode:

Constant	Value	Description
TRIGMD_FREERUN	0	There is no trigger mode. The scope triggers when it is ready to trigger, regardless of the circumstances. This is the default.
TRIGMD_SOFTWARE	1	Only user intervention can trigger the scope. No other triggering is possible.

## xPCScopes.ScopeSetTriggerMode

---

Constant	Value	Description
TRIGMD_SIGNAL	2	The scope is triggered only after a signal has crossed a value.
TRIGMD_SCOPE	3	The scope is triggered by another scope at the trigger point of the triggering scope, modified by the value of <code>triggerscopesample</code> (see <code>scopedata</code> ).

# xPCScopes.ScopeSetTriggerSample

---

<b>Purpose</b>	Set sample number for triggering scope				
<b>Prototype</b>	<code>long ScopeSetTriggerSample(long scNum, long trigScSample);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>trigScSample</i></td><td>Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>trigScSample</i>	Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>trigScSample</i>	Enter a nonnegative integer for the number of samples acquired by the triggering scope before starting data acquisition on a second scope.				
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.				
<b>Description</b>	<p>The <code>xPCScopes.ScopeSetTriggerSample</code> method sets the number of samples (<i>trigScSample</i>) a triggering scope acquires before it triggers a second scope (<i>scNum</i>). Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.</p> <p>For meaningful results, set <i>trigScSample</i> between -1 and (<i>nSamp</i>-1). <i>nSamp</i> is the number of samples in one data acquisition cycle for the triggering scope. If you specify too large a value, the scope is never triggered.</p> <p>If you want to trigger a second scope at the end of a data acquisition cycle for the triggering scope, use a value of -1 for <i>trigScSamp</i>.</p>				

# xPCScopes.ScopeSetTriggerSignal

---

**Purpose** Select signal to trigger scope

**Prototype** `long ScopeSetTriggerSignal(long scNum, long triggerSignal);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
[in] <i>triggerSignal</i>	Enter a signal number.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeSetTriggerSignal` method sets the trigger signal of scope *scNum* to *trigSig*. The trigger signal *trigSig* must be one of the signals in the scope. Use this method only when the scope is stopped. You can use `xPCScopes.ScopeGetSignals` to get the list of signals in the scope. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

# xPCScopes.ScopeSetTriggerSlope

---

**Purpose** Set slope of signal that triggers scope

**Prototype** `long ScopeSetTriggerSlope(long scNum, long triggerSlope);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

- [in] *scNum* Enter the scope number.
- [in] *triggerSlope* Enter the slope mode for the signal that triggers the scope.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeSetTriggerSlope` method sets the trigger slope of scope *scNum* to *triggerSlope*. Use this method only when the scope is stopped. Use `xPCScopes.ScopeGetState` to check the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

Use the constants defined here to set the trigger slope:

Constant	Value	Description
TRIGSLOPE_EITHER	0	The trigger slope can be either rising or falling.
TRIGSLOPE_RISING	1	The trigger signal value must be rising when it crosses the trigger value.
TRIGSLOPE_FALLING	2	The trigger signal value must be falling when it crosses the trigger value.



<b>Purpose</b>	Set software trigger of scope
<b>Prototype</b>	<code>long ScopeSoftwareTrigger(long scNum);</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes
<b>Arguments</b>	[in] <i>scNum</i> Enter the scope number.
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	<p>The <code>xPCScopes.ScopeSoftwareTrigger</code> method triggers scope <i>scNum</i>. The scope must be in the state <code>Waiting for trigger</code> for this method to succeed. Use <code>xPCScopes.ScopeGetState</code> to check the state of the scope. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.</p> <p>You can use the <code>xPCScopes.ScopeSoftwareTrigger</code> method to trigger the scope, regardless of the trigger mode.</p>

# xPCScopes.ScopeStart

---

**Purpose** Start data acquisition for scope

**Prototype** long ScopeStart(long *scNum*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The xPCScopes.ScopeStart method starts or restarts the data acquisition of scope *scNum*. If the scope does not have to preacquire samples, it enters the `Waiting for Trigger` state. The scope must be in state `Waiting to Start`, `Finished`, or `Interrupted` for this function to succeed. Call xPCScopes.ScopeGetState to check the state of the scope or, for host scopes that are already started, call xPCScopes.IsScopeFinished. Use the xPCScopes.GetScopes method to get a list of scopes.

**Purpose** Stop data acquisition for scope

**Prototype** `long ScopeStop(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCScopes.ScopeStop` method stops the scope *scNum*. This sets the scope to the `Interrupted` state. The scope must be running for this function to succeed. Use `xPCScopes.ScopeGetState` to determine the state of the scope. Use the `xPCScopes.GetScopes` method to get a list of scopes.

# xPCScopes.TargetScopeGetGrid

---

**Purpose** Get status of grid line for particular scope

**Prototype** `long TargetScopeGetGrid(long scNum);`

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The `xPCScopes.TargetScopeGetGrid` method returns the state of the grid lines for scope *scNum*. If the method detects an error, it returns -1.

**Description** The `xPCScopes.TargetScopeGetGrid` method gets the state of the grid lines for scope *scNum* (which must be of type `SCTYPE_TARGET`). A return value of 1 implies grid on, while 0 implies grid off. Note that when the scope mode is set to `SCMODE_NUMERICAL`, the grid is not drawn even when the `grid` mode is set to 1.

---

## Tip

- Use the `xPCScopes.GetScopes` method to get a list of scopes.
  - Use `xPCScopes.TargetScopeGetMode` and `xPCScopes.TargetScopeSetMode` to retrieve and set the scope mode.
-

<b>Purpose</b>	Get scope mode for displaying signals		
<b>Prototype</b>	<code>long TargetScopeGetMode(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td><code>[in] <i>scNum</i></code></td><td>Enter the scope number.</td></tr></table>	<code>[in] <i>scNum</i></code>	Enter the scope number.
<code>[in] <i>scNum</i></code>	Enter the scope number.		
<b>Return</b>	<p>The <code>xPCScopes.TargetScopeGetMode</code> method returns the value corresponding to the scope mode. The possible values are</p> <ul style="list-style-type: none"><li>• <code>SCMODE_NUMERICAL = 0</code></li><li>• <code>SCMODE_REDRAW = 1</code></li><li>• <code>SCMODE_SLIDING = 2</code></li><li>• <code>SCMODE_ROLLING = 3</code></li></ul> <p>If the method detects an error, it returns -1.</p>		
<b>Description</b>	The <code>xPCScopes.TargetScopeGetMode</code> method gets the mode of the scope <code>scNum</code> , which must be of type <code>SCTYPE_TARGET</code> . Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.		
<b>See Also</b>	API function <code>xPCScopes.TargetScopeGetModeStr</code>		

# xPCScopes.TargetScopeGetModeStr

---

**Purpose** Get scope mode string for displaying signals

**Prototype** BSTR TargetScopeGetModeStr(long *scNum*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments** [in] *scNum* Enter the scope number.

**Return** The xPCScopes.TargetScopeGetModeStr method returns the string corresponding to the scope mode. The possible strings are

- Numerical
- Redraw
- Sliding
- Rolling

**Description** The xPCScopes.TargetScopeGetModeStr method gets the mode string of the scope *scNum*, which must be of type SCTYPE\_TARGET. Use the xPCScopes.GetScopes method to get a list of scopes.

**See Also** API function xPCScopes.TargetScopeGetMode

# xPCScopes.TargetScopeGetViewMode

---

<b>Purpose</b>	Get view mode for target computer display
<b>Prototype</b>	<code>long TargetScopeGetViewMode();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes
<b>Return</b>	The <code>xPCScopes.TargetScopeGetViewMode</code> method returns the view mode for the target computer screen. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCScopes.TargetScopeGetViewMode</code> method gets the view (zoom) mode for the target computer display. If the returned value is not zero, the number is of the scope currently displayed on the screen. If the value is 0, then all defined scopes are displayed on the target computer screen, but no scopes are in focus (all scopes are unzoomed).

# xPCScopes.TargetScopeGetYLimits

---

<b>Purpose</b>	Get <i>y</i> -axis limits for scope
<b>Prototype</b>	VARIANT TargetScopeGetYLimits(long <i>scNum</i> );
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes
<b>Arguments</b>	[in] <i>scNum</i> Enter the scope number.
<b>Return</b>	The xPCScopes.TargetScopeGetYLimits method returns the upper and lower limits for target scopes.
<b>Description</b>	The xPCScopes.TargetScopeGetYLimits method gets and copies the upper and lower limits for a scope of type SCTYPE_TARGET and with scope number <i>scNum</i> . If both elements are zero, the limits are autoscaled. Use the xPCScopes.GetScopes method to get a list of scopes.



<b>Purpose</b>	Set grid mode for scope				
<b>Prototype</b>	<code>long TargetScopeSetGrid(long <i>scNum</i>, long <i>gridonoff</i>);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in] <i>gridonoff</i></td><td>Enter a grid value (0 or 1).</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in] <i>gridonoff</i>	Enter a grid value (0 or 1).
[in] <i>scNum</i>	Enter the scope number.				
[in] <i>gridonoff</i>	Enter a grid value (0 or 1).				
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.				
<b>Description</b>	The <code>xPCScopes.TargetScopeSetGrid</code> method sets the grid of a scope of type <code>SCTYPE_TARGET</code> and scope number <i>scNum</i> to <i>gridonoff</i> . If <i>gridonoff</i> is 0, the grid is off. If <i>gridonoff</i> is 1, the grid is on and grid lines are drawn on the scope window. When the drawing mode of scope <i>scNum</i> is set to <code>SCMODE_NUMERICAL</code> , the grid is not drawn even when the grid mode is set to 1. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.				

# xPCScopes.TargetScopeSetMode

---

**Purpose** Set display mode for scope

**Prototype** long TargetScopeSetMode(long *scNum*, long *mode*);

**Member Of** XPCAPICOMLIB.xPCScopes

**Arguments**

[in] <i>scNum</i>	Enter the scope number.
in] <i>mode</i>	Enter the value for the mode.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The xPCScopes.TargetScopeSetMode method sets the mode of a scope of type SCTYPE\_TARGET and scope number *scNum* to *mode*. You can use one of the following constants for *mode*:

- SCMODE\_NUMERICAL = 0
- SCMODE\_REDRAW = 1
- SCMODE\_SLIDING = 2
- SCMODE\_ROLLING = 3

Use the xPCScopes.GetScopes method to get a list of scopes.

# xPCScopes.TargetScopeSetViewMode

---

<b>Purpose</b>	Set view mode for scope		
<b>Prototype</b>	<code>long TargetScopeSetViewMode(long <i>scNum</i>);</code>		
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes		
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.
[in] <i>scNum</i>	Enter the scope number.		
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.		
<b>Description</b>	The <code>xPCScopes.TargetScopeSetViewMode</code> method sets the target computer screen to display one scope with scope number <i>scNum</i> . If you set <i>scNum</i> to 0, the target computer screen displays all the defined scopes. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.		

# xPCScopes.TargetScopeSetYLimits

---

<b>Purpose</b>	Set <i>y</i> -axis limits for scope				
<b>Prototype</b>	<code>long TargetScopeSetYLimits(long <i>scNum</i>, SAFEARRAY(double)* <i>YLimitarray</i>);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCScopes				
<b>Arguments</b>	<table><tr><td>[in] <i>scNum</i></td><td>Enter the scope number.</td></tr><tr><td>[in, out] <i>YLimitarray</i></td><td>Enter a two-element array.</td></tr></table>	[in] <i>scNum</i>	Enter the scope number.	[in, out] <i>YLimitarray</i>	Enter a two-element array.
[in] <i>scNum</i>	Enter the scope number.				
[in, out] <i>YLimitarray</i>	Enter a two-element array.				
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.				
<b>Description</b>	The <code>xPCScopes.TargetScopeSetYLimits</code> method sets the <i>y</i> -axis limits for a scope with scope number <i>scNum</i> and type <code>SCTYPE_TARGET</code> to the values in the double array <i>YLimitArray</i> . The first element is the lower limit, and the second element is the upper limit. Set both limits to 0.0 to specify autoscaling. Use the <code>xPCScopes.GetScopes</code> method to get a list of scopes.				

<b>Purpose</b>	Get average task execution time
<b>Prototype</b>	<code>double AverageTET();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCTarget</code>
<b>Return</b>	The <code>xPCTarget.AverageTET</code> method returns the average task execution time (TET) for the target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.AverageTET</code> method gets the TET for the target application. You can use this function when the target application is running or when it is stopped.

# xPCTarget.GetAppName

---

<b>Purpose</b>	Get target application name
<b>Prototype</b>	BSTR GetAppName();
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The xPCTarget.GetAppName method returns a string with the name of the target application.
<b>Description</b>	The xPCTarget.GetAppName method gets the name of the target application. You can use the return value, <i>model_name</i> , in a printf or similar statement. In case of error, the string is unchanged. Be sure to allocate enough space to accommodate the longest target name you have.

<b>Purpose</b>	Get execution time for target application
<b>Prototype</b>	<code>double GetExecTime();</code>
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The <code>xPCTarget.GetExecTime</code> method returns the current execution time for a target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetExecTime</code> method gets the current execution time for the running target application. If the target application is stopped, the value is the last running time when the target application was stopped. If the target application is running, the value is the current running time.

## xPCTarget.GetNumOutputs

---

<b>Purpose</b>	Get number of outputs
<b>Prototype</b>	<code>long GetNumOutputs();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCTarget</code>
<b>Return</b>	The <code>xPCTarget.GetNumOutputs</code> method returns the number of outputs in the current target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetNumOutputs</code> method gets the number of outputs in the target application. The number of outputs equals the sum of the input signal widths of the output blocks at the root level of the Simulink model.



<b>Purpose</b>	Get number of tunable parameters
<b>Prototype</b>	<code>long GetNumParams();</code>
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The <code>xPCTarget.GetNumParams</code> method returns the number of tunable parameters in the target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetNumParams</code> method gets the number of tunable parameters in the target application. Use this method to see how many parameters you can get or modify.

## xPCTarget.GetNumSignals

---

<b>Purpose</b>	Get number of signals
<b>Prototype</b>	<code>long GetNumSignals();</code>
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The <code>xPCTarget.GetNumSignals</code> method returns the number of signals in the target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetNumSignals</code> method gets the total number of signals in the target application that can be monitored from the host. Use this method to see how many signals you can monitor.

<b>Purpose</b>	Get number of states
<b>Prototype</b>	<code>long GetNumStates();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCTarget</code>
<b>Return</b>	The <code>xPCTarget.GetNumStates</code> method returns the number of states in the target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetNumStates</code> method gets the number of states in the target application.

# xPCTarget.GetOutputLog

---

**Purpose** Copy output log data to array

**Prototype** VARIANT GetOutputLog(long *start*, long *numsamples*, long *decimation*, long *output\_id*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments**

[in] <i>start</i>	Enter the index of the first sample to copy.
[in] <i>numsamples</i>	Enter the number of samples to copy from the output log.
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[in] <i>output_id</i>	Enter an output identification number.

**Return** The xPCTarget.GetOutputLog method returns output log data. You get the data for each output signal. If the method detects an error, it returns VT\_ERROR, a scalar.

**Description** The xPCTarget.GetOutputLog method gets the output log and copies that log to an array. Output IDs range from 0 to (N-1), where N is the return value of xPCTarget.GetNumOutputs. Entering 1 for *decimation* copies all values. Entering N copies every Nth value.

For *start*, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Get the maximum number of samples by calling the method xPCTarget.NumLogSamples.

Note that the target application must be stopped before you get the output log data.

<b>Purpose</b>	Get parameter values
<b>Prototype</b>	VARIANT GetParam(long <i>paramIdx</i> );
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Arguments</b>	[in] <i>paramIdx</i> Enter the index for a parameter.
<b>Return</b>	The xPCTarget.GetParam method returns the parameter values of a parameter.
<b>Description</b>	The xPCTarget.GetParam method gets the parameter values of a parameter identified by <i>paramIdx</i> . This method returns an array of type VARIANT containing the parameter values, with the conversion of the values being done in column-major format. Each element in the array is a double, regardless of the data type of the actual parameter. You can query the dimensions of the array by calling the method xPCTarget.GetParamDims. See the Microsoft Visual Basic® .NET 2003 solution located in C:\matlabroot\toolbox\rtw\targets\xpc\api\VBNET\SigsAndParamsDemo for an example of how to use this method.
<b>See Also</b>	API method xPCTarget.GetParamDims, xPCTarget.SetParam

# xPCTarget.GetParamDims

---

**Purpose** Get row and column dimensions of parameter

**Prototype** VARIANT GetParamDims(long *paramIdx*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *paramIdx* Parameter index.

**Return** The xPCTarget.GetParamDims method returns a VARIANT array of two elements.

**Description** The xPCTarget.GetParamDims method gets a VARIANT array of two elements. The first element contains the number of rows of the parameter, the second element contains the number of columns for your parameter.

<b>Purpose</b>	Get parameter index				
<b>Prototype</b>	<code>long GetParamIdx(BSTR <i>blockName</i>, BSTR <i>paramName</i>);</code>				
<b>Member Of</b>	XPCAPICOMLib.xPCTarget				
<b>Arguments</b>	<table><tr><td>[in] <i>blockName</i></td><td>Enter the full block path generated by the Simulink Coder software.</td></tr><tr><td>[in] <i>paramName</i></td><td>Enter the parameter name for a parameter associated with the block.</td></tr></table>	[in] <i>blockName</i>	Enter the full block path generated by the Simulink Coder software.	[in] <i>paramName</i>	Enter the parameter name for a parameter associated with the block.
[in] <i>blockName</i>	Enter the full block path generated by the Simulink Coder software.				
[in] <i>paramName</i>	Enter the parameter name for a parameter associated with the block.				
<b>Return</b>	The <code>xPCTarget.GetParamIdx</code> method returns the parameter index for the parameter name. If the method detects an error, it returns -1.				
<b>Description</b>	The <code>xPCTarget.GetParamIdx</code> method gets the parameter index for the parameter name ( <i>paramName</i> ) associated with a Simulink block ( <i>blockName</i> ). Both <i>blockName</i> and <i>paramName</i> must be identical to those generated at target application building time. The block names should be referenced from the file <i>model_namept.m</i> in the generated code, where <i>model_name</i> is the name of the model. Note that a block can have one or more parameters.				

# xPCTarget.GetParamName

---

**Purpose** Get parameter name

**Prototype** VARIANT GetParamName(long *paramIdx*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *paramIdx* Enter a parameter index.

**Return** The xPCTarget.GetParamName method returns a VARIANT array that contains two elements, the block path and parameter name, as strings.

**Description** The xPCTarget.GetParamName method gets the parameter name and block name for a parameter with the index *paramIdx*. If *paramIdx* is invalid, xPCGetLastError returns nonzero, and the strings are unchanged. Get the parameter index with the method xPCTarget.GetParamIdx.



<b>Purpose</b>	Get sample time
<b>Prototype</b>	<code>double GetSampleTime();</code>
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The <code>xPCTarget.GetSampleTime</code> method returns the sample time, in seconds, of the target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetSampleTime</code> method gets the sample time, in seconds, of the target application. You can get the error by using the method <code>xPCGetLastError</code> .

# xPCTarget.GetSignal

---

**Purpose** Get signal value

**Prototype** `double GetSignal(long sigNum);`

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *sigNum* Enter a signal number.

**Return** The xPCTarget.GetSignal method returns the current value of signal *sigNum*. If the method detects an error, it returns -1.

**Description** The xPCTarget.GetSignal method gets the current value of a signal. Use the xPCTarget.GetSignalIdx method to get the signal number.

# xPCTarget.GetSignalidsfromLabel

---

<b>Purpose</b>	Get signal IDs from signal label
<b>Prototype</b>	VARIANT GetSignalidsfromLabel(BSTR <i>sigLabel</i> );
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Arguments</b>	[in] <i>sigLabel</i> Enter a signal label.
<b>Return</b>	The xPCTarget.GetSignalidsfromLabel method returns a VARIANT array of the signal elements contained in the signal <i>sigLabel</i> . If no labels exist, the method returns an empty string.
<b>Description</b>	<p>The xPCTarget.GetSignalidsfromLabel method returns a VARIANT array of the signal elements contained in the signal <i>sigLabel</i>. Signal labels must be unique.</p> <p>This method assumes that you have labeled the signal for which you request the indices (see the <b>Signal name</b> parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.</p>
<b>See Also</b>	API method xPCTarget.GetSignalLabel

# xPCTarget.GetSignalLabel

---

**Purpose** Get signal label

**Prototype** BSTR GetSignalLabel(long *sigIdx*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *sigIdx* Enter a signal index.

**Return** The xPCTarget.GetSignalLabel method returns the label of the signal. If no labels exist, the method returns an empty string.

**Description** The xPCTarget.GetSignalLabel method copies and gets the signal label of a signal with *sigIdx*. The method returns the signal label. This method assumes that you already know the signal index. Signal labels must be unique.

This method assumes that you have labeled the signal for which you request the indices (see the **Signal name** parameter of the “Signal Properties Controls”). Note that the xPC Target software refers to Simulink signal names as signal labels. The creator of the application should already know the signal name/label.

**See Also** API method xPCTarget.GetSignalIdsfromLabel

**Purpose** Get signal index

**Prototype** `long GetSignalIdx(BSTR sigName);`

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *sigName* Enter a signal name.

**Return** The xPCTarget.GetSignalIdx method returns the index for the signal with name *sigName*. If the method detects an error, it returns -1.

**Description** The xPCTarget.GetSignalIdx method gets the index of a signal. The name must be identical to the name generated when the application was built. You should reference the name from the file *model\_namebio.m* in the generated code, where *model\_name* is the name of the model. The creator of the application should already know the signal name.

# xPCTarget.GetSignalName

---

**Purpose** Copy signal name to character array

**Prototype** BSTR GetSignalName(long *sigIdx*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *sigIdx* Enter a signal index.

**Return** The xPCTarget.GetSignalName method returns the name of the signal.

**Description** The xPCTarget.GetSignalName method copies and gets the signal name, including the block path, of a signal with *sigIdx*. The method returns a signal name, which makes it convenient to use in a `printf` or similar statement. This method assumes that you already know the signal index.

**Purpose** Get vector of signal values

**Prototype** `VARIANT GetSignals(long NumOfSignals, SAFEARRAY(int)* SignalsIdxArray);`

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments**

[in] <i>NumOfSignals</i>	Enter the number of signals to acquire (the number of IDs in <i>SignalsIdxArray</i> ).
[out] <i>SignalsIdxArray</i>	Enter the IDs of the signals to acquire.

**Return** The `xPCTarget.GetSignals` method returns a double-valued variant array containing the current value of a vector of signals. If the method detects an error, it returns `VT_ERROR`, a scalar.

**Description** This function returns the values of a vector of up to 1000 signals as fast as it can acquire them. The values are converted to doubles regardless of the actual data type of the signal.

---

### Tip

- Pass an integer array of signal numbers into *SignalsIdxArray*. Get the signal numbers with the function `xPCTarget.GetSignalIdx`.
  - The signal values may not be at the same time step. To get signal values at the same time step, define a scope of type `SCTYPE_HOST` and use `xPCScopes.ScopeGetData`.
- 

The function `xPCTarget.GetSignal` does the same thing for a single signal, and could be used multiple times to achieve the same result.

## xPCTarget.GetSignals

---

However, xPCGetSignals is faster and the signal values are more likely to be spaced closely together.

### **See Also**

API functions `xPCTarget.GetSignal`, `xPCTarget.GetSignalIdx`



**Purpose** Get width of signal

**Prototype** `long GetSignalWidth(long sigIdx);`

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments** [in] *sigIdx* Enter the index of a signal.

**Return** The `xPCTarget.GetSignalWidth` method returns the signal width for a signal with *sigIdx*. If the method detects an error, it returns -1.

**Description** The `xPCTarget.GetSignalWidth` method gets the number of signals for a specified signal index. Although signals are manipulated as scalars, the width of the signal might be useful to reassemble the components into a vector. A signal's width is the number of signals in the vector.

# xPCTarget.GetStateLog

---

**Purpose** Get state log

**Prototype** VARIANT GetStateLog(long *start*, long *numsamples*, long *decimation*, long *state\_id*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments**

[in] <i>start</i>	Enter the index of the first sample to copy.
[in] <i>numsamples</i>	Enter the number of samples to copy from the output log.
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[in] <i>state_id</i>	Enter a state identification number.
[out, retval] <i>Outarray</i>	The log is stored in <i>Outarray</i> , whose allocation is the responsibility of the caller.

**Return** The xPCTarget.GetStateLog method returns the state log. If the method detects an error, it returns VT\_ERROR, a scalar.

**Description** The xPCTarget.GetStateLog method gets the state log. You get the data for each state signal in turn by specifying the *state\_id*. State IDs range from 1 to (N-1), where N is the return value of xPCTarget.GetNumStates. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *start*, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Use the xPCTarget.NumLogSamples method to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

<b>Purpose</b>	Get stop time
<b>Prototype</b>	<code>double GetStopTime();</code>
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The <code>xPCTarget.GetStopTime</code> method returns the stop time as a double, in seconds, of the target application. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.GetStopTime</code> method gets the stop time, in seconds, of the target application. This is the amount of time the target application runs before stopping.

# xPCTarget.GetTETLog

---

**Purpose** Get TET log

**Prototype** VARIANT GetTETLog(long *start*, long *numsamples*, long *decimation*);

**Member Of** XPCAPICOMLib.xPCTarget

**Arguments**

[in] <i>start</i>	Enter the index of the first sample to copy.
[in] <i>numsamples</i>	Enter the number of samples to copy from the TET log.
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[out, retval] <i>Outarray</i>	The log is stored in <i>Outarray</i> , whose allocation is the responsibility of the caller.

**Return** The xPCTarget.GetTETLog method returns the TET log. If the method detects an error, it returns VT\_ERROR, a scalar.

**Description** The xPCTarget.GetTETLog method gets the task execution time (TET) log. Entering 1 for *decimation* copies all values. Entering N copies every Nth value. For *start*, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Use the xPCTarget.NumLogSamples method to get the maximum number of samples.

Note that the target application must be stopped before you get the number.

<b>Purpose</b>	Get time log						
<b>Prototype</b>	VARIANT GetTimeLog(long <i>start</i> , long <i>numsamples</i> , long <i>decimation</i> );						
<b>Member Of</b>	XPCAPICOMLib.xPCTarget						
<b>Arguments</b>	<table><tr><td>[in] <i>start</i></td><td>Enter the index of the first sample to copy.</td></tr><tr><td>[in] <i>numsamples</i></td><td>Enter the number of samples to copy from the time log.</td></tr><tr><td>[in] <i>decimation</i></td><td>Select whether to copy all the sample values or every Nth value.</td></tr></table>	[in] <i>start</i>	Enter the index of the first sample to copy.	[in] <i>numsamples</i>	Enter the number of samples to copy from the time log.	[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.
[in] <i>start</i>	Enter the index of the first sample to copy.						
[in] <i>numsamples</i>	Enter the number of samples to copy from the time log.						
[in] <i>decimation</i>	Select whether to copy all the sample values or every Nth value.						
<b>Return</b>	The xPCTarget.GetTimeLog method returns the time log. If the method detects an error, it returns VT_ERROR, a scalar.						
<b>Description</b>	<p>The xPCTarget.GetTimeLog method gets the time log. This is especially relevant in the case of value-equidistant logging, where the logged values might not be uniformly spaced in time. Entering 1 for <i>decimation</i> copies all values. Entering N copies every Nth value. For <i>start</i>, the sample indices range from 0 to (N-1), where N is the return value of xPCTarget.NumLogSamples. Use the xPCTarget.NumLogSamples method to get the number of samples.</p> <p>Note that the target application must be stopped before you get the number.</p>						

# xPCTarget.GetxPCError

---

<b>Purpose</b>	Get error string
<b>Prototype</b>	BSTR GetxPCError();
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	The xPCTarget.GetxPCError method returns the string for the last reported error. If the software has not reported an error, this method returns 0.
<b>Description</b>	The xPCTarget.GetxPCError method gets the string of the error last reported by another COM API method. This value is reset every time you call a new method. Therefore, you should check this constant value immediately after a call to an API COM method. You can use this method in conjunction with the xPCTarget.isxPCError method, which detects that an error has occurred.
<b>See Also</b>	API method xPCTarget.isxPCError

<b>Purpose</b>	Initialize target object to communicate with target computer
<b>Prototype</b>	<code>long Init(IxPCProtocol* xPCProtocol);</code>
<b>Member Of</b>	XPCAPICOMLib.xPCTarget
<b>Return</b>	<p>If the method detects an error, it returns -1. Otherwise, it returns 0.</p> <p>If the xPCTarget.Init method initializes the target object without detecting an error, it returns 0. If the target object fails to initialize, this method returns -1.</p>
<b>Description</b>	The xPCTarget.Init method initializes the target object to communicate with the target computer referenced by the xPCProtocol object.

# xPCTarget.IsAppRunning

---

<b>Purpose</b>	Return running status for target application
<b>Prototype</b>	<code>long IsAppRunning();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCTarget</code>
<b>Return</b>	If the target application is stopped, the <code>xPCTarget.IsAppRunning</code> method returns 0. If the target application is running, this method returns 1. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.IsAppRunning</code> method returns 1 or 0 depending on whether the target application is stopped or running.



<b>Purpose</b>	Return overload status for target computer
<b>Prototype</b>	<code>long IsOverloaded();</code>
<b>Member Of</b>	<code>XPCAPICOMLib.xPCTarget</code>
<b>Return</b>	If the target application has overloaded the CPU, the <code>xPCTarget.IsOverloaded</code> method returns 1. If it has not overloaded the CPU, the method returns 0. If the method detects an error, it returns -1.
<b>Description</b>	The <code>xPCTarget.IsOverloaded</code> method checks if the target application has overloaded the target computer and returns 1 if it has and 0 if it has not. If the target application is not running, the method returns 0.

# xPCTarget.isxPCError

---

<b>Purpose</b>	Return error status
<b>Prototype</b>	<code>long isxPCError();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	If an error occurred, the method returns 1. Otherwise, it returns 0.
<b>Description</b>	Use the <code>xPCTarget.isxPCError</code> method to check for errors that might occur after a call to the <code>xPCTarget</code> class methods. If the method detects that an error occurred, call the <code>xPCTarget.GetxPCError</code> method to get the string for the error.
<b>See Also</b>	API method <code>xPCTarget.GetxPCError</code>

**Purpose** Load target application onto target computer

**Prototype** long LoadApp(BSTR *pathstr*, BSTR *filename*);

**Member Of** XPCAPICOMLIB.xPCTarget

**Arguments**

[in] <i>pathstr</i>	Enter the full path to the target application file, excluding the file name. For example, in C, use a string like "C:\\work", in Microsoft Visual Basic, use a string like 'C:\\work'.
[in] <i>filename</i>	Enter the name of a compiled target application (*.dlm) without the file extension. For example, in C use a string like "xpcosc", in Microsoft Visual Basic, use a string like 'xpcosc'.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The xPCTarget.LoadApp method loads the compiled target application to the target computer. *pathstr* must not contain the trailing backslash. *pathstr* can be set to NULL or to the string 'nopath' if the application is in the current folder. The variable *filename* must not contain the target application extension.

Before returning, xPCTarget.LoadApp waits for a certain amount of time before checking whether the model initialization is complete. In the case where the model initialization is incomplete, xPCTarget.LoadApp returns a timeout error to indicate a connection problem (for example, ETCPREAD). By default, xPCTarget.LoadApp checks for target readiness five times, with each attempt taking approximately 1 second (less if the target is ready). However, for larger models or models requiring longer initialization (for example, those with thermocouple boards), the default might not be long enough and a spurious timeout can

## xPCTarget.LoadApp

---

be generated. The methods `xPCProtocol.GetLoadTimeOut` and `xPCProtocol.SetLoadTimeOut` control the number of attempts made.

<b>Purpose</b>	Copy maximum task execution time to array
<b>Prototype</b>	VARIANT MaximumTET();
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	The xPCTarget.MaximumTET method returns a VARIANT object containing the maximum task execution time (TET) and the time at which the maximum TET was achieved. The maximum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.
<b>Description</b>	The xPCTarget.MaximumTET method returns the maximum TET that was achieved during the previous target application run.

# xPCTarget.MaxLogSamples

---

<b>Purpose</b>	Return maximum number of samples that can be in log buffer
<b>Prototype</b>	<code>long MaxLogSamples();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	The <code>xPCTarget.MaxLogSamples</code> method returns the total number of samples. If the method detects an error, it returns -1.
<b>Description</b>	<p>The <code>xPCTarget.MaxLogSamples</code> method returns the total number of samples that can be returned in the logging buffers.</p> <p>Note that the target application must be stopped before you get the number.</p>

<b>Purpose</b>	Copy minimum task execution time to array
<b>Prototype</b>	VARIANT MinimumTET();
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	The xPCTarget.MinimumTET method returns a VARIANT object containing the minimum task execution time (TET) and the time at which the minimum TET was achieved. The minimum TET value is copied into the first element, and the time at which it was achieved is copied into the second element.
<b>Description</b>	The xPCTarget.MinimumTET method returns the minimum task execution time (TET) that was achieved during the previous target application run.

# xPCTarget.NumLogSamples

---

<b>Purpose</b>	Return number of samples in log buffer
<b>Prototype</b>	<code>long NumLogSamples();</code>
<b>Member Of</b>	<code>XPCAPICOMLIB.xPCTarget</code>
<b>Return</b>	The <code>xPCTarget.NumLogSamples</code> method returns the number of samples in the log buffer. If the method detects an error, it returns -1.
<b>Description</b>	<p>The <code>xPCTarget.NumLogSamples</code> method returns the number of samples in the log buffer. In contrast to <code>xPCTarget.MaxLogSamples</code>, which returns the maximum number of samples that can be logged (because of buffer size constraints), <code>xPCTarget.NumLogSamples</code> returns the number of samples actually logged.</p> <p>Note that the target application must be stopped before you get the number.</p>



<b>Purpose</b>	Return number of times log buffer wraps
<b>Prototype</b>	<code>long NumLogWraps();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	The <code>xPCTarget.NumLogWraps</code> method returns the number of times the log buffer wraps. If the method detects an error, it returns -1.
<b>Description</b>	<p>The <code>xPCTarget.NumLogWraps</code> method returns the number of times the log buffer wraps.</p> <p>Note that the target application must be stopped before you get the number.</p>

# xPCTarget.SetParam

---

<b>Purpose</b>	Change parameter value				
<b>Prototype</b>	<code>long SetParam(long <i>paramIdx</i>, SAFEARRAY(double)* <i>newparamVal</i>);</code>				
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget				
<b>Arguments</b>	<table><tr><td>[in] <i>paramIdx</i></td><td>Parameter index.</td></tr><tr><td>[in, out] <i>newparamVal</i></td><td>Vector of doubles, assumed to be the size required by the parameter type.</td></tr></table>	[in] <i>paramIdx</i>	Parameter index.	[in, out] <i>newparamVal</i>	Vector of doubles, assumed to be the size required by the parameter type.
[in] <i>paramIdx</i>	Parameter index.				
[in, out] <i>newparamVal</i>	Vector of doubles, assumed to be the size required by the parameter type.				
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.				
<b>Description</b>	The <code>xPCTarget.SetParam</code> method sets the parameter <i>paramIdx</i> to the value in <i>newparamVal</i> . For matrices, <i>newparamVal</i> should be a vector representation of the matrix in column-major format. Although <i>newparamVal</i> is a vector of doubles, the method converts the values to the expected data types (using truncation) before setting them.				
<b>See Also</b>	API methods <code>xPCTarget.GetParam</code> , <code>xPCTarget.GetParamDims</code> , <code>xPCTarget.GetParamIdx</code>				

<b>Purpose</b>	Change sample time for target application
<b>Prototype</b>	<code>long SetSampleTime(double ts);</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Arguments</b>	[in] <i>ts</i> Sample time for the target application.
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	The <code>xPCTarget.SetSampleTime</code> method sets the sample time, in seconds, of the target application to <i>ts</i> . Use this method only when the application is stopped.

# xPCTarget.SetStopTime

---

**Purpose** Change stop time of target application

**Prototype** `long SetStopTime(double tfinal);`

**Member Of** XPCAPICOMLIB.xPCTarget

**Arguments** [in] *tfinal* Enter the stop time, in seconds.

**Return** If the method detects an error, it returns 0. Otherwise, it returns -1.

**Description** The `xPCTarget.SetStopTime` method sets the stop time of the target application to the value in *tfinal*. The target application will run for this number of seconds before stopping. Set *tfinal* to -1.0 to set the stop time to infinity.

<b>Purpose</b>	Start target application
<b>Prototype</b>	long StartApp()
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	The xPCTarget.StartApp method starts the target application loaded on the target machine.

# xPCTarget.StopApp

---

<b>Purpose</b>	Stop target application
<b>Prototype</b>	<code>long StopApp();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	The <code>xPCTarget.StopApp</code> method stops the target application loaded on the target computer. The target application remains loaded, and the parameter changes you made remain intact. If you want to stop and unload an application, use <code>xPCTarget.UnLoadApp</code> .

<b>Purpose</b>	Unload target application
<b>Prototype</b>	<code>long UnLoadApp();</code>
<b>Member Of</b>	XPCAPICOMLIB.xPCTarget
<b>Return</b>	If the method detects an error, it returns 0. Otherwise, it returns -1.
<b>Description</b>	The <code>xPCTarget.UnLoadApp</code> method stops the current target application, removes it from the target computer memory, and resets the target computer in preparation for receiving a new target application. The method <code>xPCTarget.LoadApp</code> calls this method before loading a new target application.

# xPCTarget.UnLoadApp

---



# Configuration Parameters

---

This topic deals with configuration parameters in xPC Target Explorer and in the MATLAB API.

## Setting Configuration Parameters

**In this section...**

“xPC Target options Pane” on page 4-3

“Automatically download application after building” on page 4-4

“Download to default target PC” on page 4-5

“Specify target PC name” on page 4-6

“Name of xPC Target object created by build process” on page 4-7

“Use default communication timeout” on page 4-8

“Specify the communication timeout in seconds” on page 4-9

“Execution mode” on page 4-10

“Real-time interrupt source” on page 4-11

“I/O board generating the interrupt” on page 4-12

“PCI slot (-1: autosearch) or ISA base address” on page 4-16

“Log Task Execution Time” on page 4-17

“Signal logging data buffer size in doubles” on page 4-18

“Enable profiling” on page 4-20

“Number of events (each uses 20 bytes)” on page 4-21

“Double buffer parameter changes” on page 4-22

“Load a parameter set from a file on the designated target file system” on page 4-24

“File name” on page 4-25

“Build COM objects from tagged signals/parameters” on page 4-26

“Generate CANape extensions” on page 4-27

“Include model hierarchy on the target application” on page 4-28

“Enable Stateflow animation” on page 4-29

## xPC Target options Pane

Set up general information about building target applications, including target, execution, data logging, and other options.

### Configuration

To enable the xPC Target options pane, you must:

- 1 Select `xpctarget.tlc` or `xpctargetert.tlc` for the **System target file** parameter on the code generation pane.
- 2 Select **C** for the **Language** parameter on the code generation pane.

### Tips

- The default xPC Target options work for the generation of most target applications. If you want to customize the build of your target application, set the option parameters to suit your specifications.
- To access these parameters from the MATLAB command line, use:
  - `gcs` — To access the current model.
  - `set_param` — To set the parameter value.
  - `get_param` — To get the current value of the parameter.

### See Also

“xPC Target Options Configuration Parameters”

### Automatically download application after building

Enable Simulink Coder to build and download the target application to the target computer.

#### Settings

**Default:** on



On

Builds and downloads the target application to the target computer.



Off

Builds the target application, but does not download it to the target computer.

#### Command-Line Information

**Parameter:** xPCisDownloadable

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

#### See Also

“Build and Download Target Application”

## Download to default target PC

Direct Simulink Coder to download the target application to the default target computer.

### Settings

**Default:** on



On

Downloads the target application to the default target computer. Assumes that you configured a default target computer through xPC Target Explorer.



Off

Enables the **Specify target PC name** field so that you can enter the target computer to which to download the target application.

### Dependency

This parameter enables **Specify target PC name**.

### Command-Line Information

**Parameter:** xPCisDefaultEnv

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

### See Also

- “Ethernet Communication Setup”
- “RS-232 Communication Setup”

### Specify target PC name

Specify a target computer name for your target application.

### Settings

''

### Tip

The target computer name appears in xPC Target Explorer as the target computer node, for example TargetPC1.

### Dependencies

This parameter is enabled by **Download to default target PC**.

### Command-Line Information

**Parameter:** xPCTargetPCEnvName

**Type:** string

**Value:** Any valid target computer

**Default:** ''

### See Also

“xPC Target Explorer”

## **Name of xPC Target object created by build process**

Enter the name of the target object created by the build process.

### **Settings**

**Default:** tg

### **Tip**

Use this name when you work with the target object through the command-line interface.

### **Command-Line Information**

**Parameter:** RL320bjectName

**Type:** string

**Value:** 'tg' | valid target object name

**Default:** 'tg'

### **See Also**

“Target Driver Objects”

### Use default communication timeout

Direct xPC Target software to wait 5 (default) seconds for the target application to be downloaded to the target computer.

#### Settings

**Default:** on



On

Waits the default amount of seconds (5) for the target application to be downloaded to the target computer.



Off

Enables the **Specify the communication timeout in seconds** field so that you can enter the maximum length of time in seconds you want to wait for a target application to be downloaded to the target computer.

#### Dependencies

This parameter enables **Specify the communication timeout in seconds**.

#### Command-Line Information

**Parameter:** xPCisModelTimeout

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

#### See Also

“Increase the Time for Downloads”



## Specify the communication timeout in seconds

Specify a timeout, in seconds, to wait for the target application to download to the target computer.

### Settings

**Default:** 5

### Tip

Enter the maximum length of time in seconds you want to allow the xPC Target software to wait for the target application to download to the target computer. If the target application is not downloaded within this time frame, the software generates an error.

### Dependencies

This parameter is enabled by **Use default communication timeout**.

### Command-Line Information

**Parameter:** xPCModelTimeoutSecs

**Type:** string

**Value:** Any valid number of seconds

**Default:** '5'

### See Also

“Increase the Time for Downloads”

### Execution mode

Specify target application execution mode.

### Settings

**Default:** Real-Time

Real-Time

Executes target application in real time.

Freerun

Runs the target application as fast as possible.

### Command-Line Information

**Parameter:** RL32ModeModifier

**Type:** string

**Value:** 'Real-Time' | 'Freerun'

**Default:** 'Real-Time'

### See Also

“Set Configuration Parameters”

## Real-time interrupt source

Select a real-time interrupt source from the I/O board.

### Settings

**Default:** Timer

Timer

Specifies that the board interrupt source is a timer.

Auto (PCI only)

Enables the xPC Target software to automatically determine the IRQ that the BIOS assigned to the board and use it.

3 to 15

Specifies that the board interrupt source is an IRQ number on the board.

### Tips

- The Auto (PCI only) option is available only for PCI boards. If you have an ISA board (PC 104 or onboard parallel port), you must set the IRQ manually.
- The xPC Target software treats PCI parallel port plug-in boards like ISA boards. For PCI parallel port plug-in boards, you must set the IRQ manually.
- Multiple boards can share the same interrupt number.

### Command-Line Information

**Parameter:** RL32IRQSourceModifier

**Type:** string

**Value:** 'Timer' | Auto (PCI only) | '3' | '4' | '5' | '6' | '7' | '8' | '9' | '10' | '11' | '12' | '13' | '14' | '15'

**Default:** 'Timer'

### See Also

“Set Configuration Parameters”

### I/O board generating the interrupt

Specify the board interrupt source.

#### Settings

**Default:** None/Other

ATI-RP-R5

Specifies that the interrupt source is an ATI-RP-R5 board.

AudioPMC+

Specifies that the interrupt source is the Bittware AudioPMC+ audio board.

Bitflow NEON

Specifies that the interrupt source is the BitFlow™ NEON video board.

CB\_CIO-CTR05

Specifies that the interrupt source is the Measurement Computing™ CIO-CTR05 board.

CB\_PCI-CTR05

Specifies that the interrupt source is the Measurement Computing PCI-CTR05 board.

Diamond\_MM-32

Specifies that the interrupt source is the Diamond Systems MM-32 board.

FastComm 422/2-PCI

Specifies that the interrupt source is the Fastcom 422/2-PCI board.

FastComm 422/2-PCI-335

Specifies that the interrupt source is the Fastcom 422/2-PCI-335 board.

FastComm 422/4-PCI-335

Specifies that the interrupt source is the Fastcom 422/4-PCI-335 board.

GE\_Fanuc (VMIC)\_PCI-5565

Specifies that the interrupt source is the GE® Fanuc VMIC PCI-5565 board.

General Standards 24DSI12

Specifies that the interrupt source is the General Standards 24DSI12 board.

**Parallel\_Port**

Specifies that the interrupt source is the parallel port of the target computer.

**Quatech DSCP-200/300**

Specifies that the interrupt source is the Quatech® DSCP-200/300 board.

**Quatech ESC-100**

Specifies that the interrupt source is the Quatech ESC-100 board.

**Quatech QSC-100**

Specifies that the interrupt source is the Quatech QSC-100 board.

**Quatech QSC-200/300**

Specifies that the interrupt source is the Quatech QSC-200/300 board.

**RTD\_DM6804**

Specifies that the interrupt source is the Real-Time Devices DM6804 board.

**SBS\_25x0\_ID\_0x100**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x100.

**SBS\_25x0\_ID\_0x101**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x101.

**SBS\_25x0\_ID\_0x102**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x102.

**SBS\_25x0\_ID\_0x103**

Specifies that the interrupt source is an SBS Technologies shared memory board associated with ID 0x103.

**Scramnet\_SC150+**

Specifies that the interrupt source is the Systran® Scramnet+ SC150 board.

**Softing\_CAN-AC2-104**

Specifies that the interrupt source is the Softing® CAN-AC2-104 board.

**Softing\_CAN-AC2-PCI**

Specifies that the interrupt source is the Softing CAN-AC2-PCI board.

- Speedgoat\_I0301**  
Specifies that the interrupt source is the Speedgoat IO301 FPGA board.
- Speedgoat\_I0302**  
Specifies that the interrupt source is the Speedgoat IO302 FPGA board.
- Speedgoat\_I0303**  
Specifies that the interrupt source is the Speedgoat IO303 FPGA board.
- Speedgoat\_I0311**  
Specifies that the interrupt source is the Speedgoat IO311 FPGA board.
- Speedgoat\_I0312**  
Specifies that the interrupt source is the Speedgoat IO312 FPGA board.
- Speedgoat\_I0313**  
Specifies that the interrupt source is the Speedgoat IO313 FPGA board.
- Speedgoat\_I0314**  
Specifies that the interrupt source is the Speedgoat IO314 FPGA board.
- Speedgoat\_I0325**  
Specifies that the interrupt source is the Speedgoat IO325 FPGA board.
- Speedgoat\_I0331**  
Specifies that the interrupt source is the Speedgoat IO331 FPGA board.
- UEI\_MF<sub>x</sub>**  
Specifies that the interrupt source is a United Electronic Industries UEI-MF series board.
- None/Other**  
Specifies that the I/O board has no interrupt source.

### Command-Line Information

**Parameter:** xPCIRQSourceBoard

**Type:** string

**Value:** 'ATI-RP-R5' |  
'AudioPMC+' |  
'Bitflow NEON' |  
'CB\_CIO-CTR05' |  
'CB\_PCI-CTR05' |  
'Diamond\_MM-32' |  
'FastComm 422/2-PCI' |

```
'FastComm 422/2-PCI-335' |  
'FastComm 422/4-PCI-335' |  
'GE_Fanuc(VMIC)_PCI-5565' |  
'General Standards 24DSI12' |  
'Parallel_Port' |  
'Quatech DSCP-200/300' |  
'Quatech ESC-100' |  
'Quatech QSC-100' |  
'Quatech QSC-200/300' |  
'RTD_DM6804' |  
'SBS_25x0_ID_0x100' |  
'SBS_25x0_ID_0x101' |  
'SBS_25x0_ID_0x102' |  
'SBS_25x0_ID_0x103' |  
'Scramnet_SC150+' |  
'Softing_CAN-AC2-104' |  
'Softing_CAN-AC2-PCI' |  
'Speedgoat_I0301' |  
'Speedgoat_I0302' |  
'Speedgoat_I0303' |  
'Speedgoat_I0311' |  
'Speedgoat_I0312' |  
'Speedgoat_I0313' |  
'Speedgoat_I0314' |  
'Speedgoat_I0325' |  
'Speedgoat_I0331' |  
'UEI_MFx' |  
'None/Other'
```

**Default:** 'None/Other'

## See Also

“Set Configuration Parameters”

### **PCI slot (-1: autosearch) or ISA base address**

Enter the slot number or base address for the I/O board generating the interrupt.

#### **Settings**

**Default:** -1

The PCI slot can be either -1 (let the xPC Target software determine the slot number) or of the form [bus, slot].

The base address is a hexadecimal number of the form 0x300.

#### **Tip**

To determine the bus and PCI slot number of the boards in the target computer, type `getxpcpci` in the MATLAB window.

#### **Command-Line Information**

**Parameter:** xPCIOIRQSlot

**Type:** string

**Value:** '-1' | hexadecimal value

**Default:** '-1'

#### **See Also**

“xPC Target Options Configuration Parameters”

“PCI Bus I/O Devices”



## Log Task Execution Time

Log task execution times to the target object property `tg.TETlog`.

### Settings

**Default:** on



On

Logs task execution times to the target object property `tg.TETlog`.



Off

Does not log task execution times to the target object property `tg.TETlog`.

### Command-Line Information

**Parameter:** RL32LogTETModifier

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'on'

### See Also

“xPC Target Options Configuration Parameters”

“Signal Logging Basics”

### Signal logging data buffer size in doubles

Enter the maximum number of sample points to save before wrapping.

#### Settings

**Default:** 100000

The maximum value for this option cannot exceed the available target computer memory, which the xPC Target software also uses to hold other items.

#### Tips

- Target applications use this buffer to store the time, states, outputs, and task execution time logs as defined in the Simulink model.
- The maximum value for this option derives from available target computer memory, which the xPC Target software also uses to hold other items. For example, in addition to signal logging data, the software also uses the target computer memory for the xPC Target kernel, target application, and scopes.

For example, assume that your model has six data items (one time, two states, two outputs, and one task execution time (TET)). If you enter a buffer size of 100000, the target object property `tg.MaxLogSamples` is calculated as  $\text{floor}(100000) / 6 = 16666$ . After saving 16666 sample points, the buffer wraps and further samples overwrite the older ones.

- If you enter a logging buffer size larger than the available RAM on the target computer, after downloading and initializing the target application, the target computer displays a message, `ERROR: allocation of logging memory failed`. To avoid this error, either install more RAM or reduce the buffer size for logging, and then reboot the target computer. To calculate the maximum buffer size you might have for your target application logs, divide the amount of available RAM on your target computer by `sizeof(double)`, or 8. Enter that value for the **Signal logging data buffer size in doubles** value.

#### Command-Line Information

**Parameter:** RL32LogBufSizeModifier

**Type:** string

**Value:** '100000' | any valid memory size

**Default:** '100000'

**See Also**

“xPC Target Options Configuration Parameters”

### Enable profiling

Enable profiling and visual presentation of target computer execution.

#### Settings

**Default:** off

On  
Profile target computer execution.

Off  
Do not profile target computer execution.

#### Tips

- Before building and downloading a model, select this check box to observe the target computer thread execution.
- If you are using multiple CPU cores on a target computer, select **Enabling profiling** to verify that the xPC Target software is actually executing on the multiple CPU cores.

---

**Tip** For more on configuring your model for concurrent execution, see “Design Considerations”.

---

#### Command-Line Information

**Parameter:** xPCTaskExecutionProfile

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

#### See Also

“Profiling Target Application Execution”

## Number of events (each uses 20 bytes)

Enter the maximum of events to log for the profiling tool.

### Settings

**Default:** 5000

The maximum number of events to be logged for the profiling tool.

### Tips

- An event is the start or end of an interrupt or iteration of the model. For example, one sample can have four events: the beginning and end of an interrupt, and the beginning and end of an iteration.
- Use this parameter in conjunction with the **Enable profiling** parameter.
- Each event contains information such as the CPU ID, model thread ID (TID), event ID, and time stamp readings. Each event occupies 20 bytes.

### Command-Line Information

**Parameter:** xPCRL32EventNumber

**Type:** string

**Value:** any valid number of events

**Default:** '5000'

### See Also

“Profiling Target Application Execution”

### Double buffer parameter changes

Use a double buffer for parameter tuning. This enables parameter tuning so that the process of changing parameters in the target application uses a double buffer.

#### Settings

**Default:** off



On

Changes parameter tuning to use a double buffer.



Off

Suppresses double buffering of parameter changes in the target application.

#### Tips

- When a parameter change request is received, the new value is compared to the old one. If the new value is identical to the old one, it is discarded, and if different, it is queued.
- At the start of execution of the next sample of the real-time task, the queued parameters are updated. This means that parameter tuning affects the task execution time (TET), and the very act of parameter tuning can cause a CPU overload error.
- Double buffering leads to a more robust parameter tuning interface, but it increases Task Execution Time (TET) and the higher probability of overloads. Under typical conditions, keep double buffering off (default).

#### Command-Line Information

**Parameter:** xpcDb1Buff

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

**See Also**

“xPC Target Options Configuration Parameters”

### Load a parameter set from a file on the designated target file system

Automatically load a parameter set from a file on the designated target computer file system.

#### Settings

**Default:** off

On  
Enable the automatic loading of a parameter set from the file specified by **File name** on the designated target computer file system.

Off  
Suppress the automatic loading of a parameter set from a file on the designated target computer file system.

#### Dependencies

This parameter enables **File name**.

#### Command-Line Information

**Parameter:** xPCLoadParamSetFile

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

#### See Also

“xPC Target Options Configuration Parameters”

“Save and Reload Parameters Using MATLAB Language”



## File name

Specify the target computer file name from which to load the parameter set.

## Settings

''

## Tip

If the named file does not exist, the software loads the parameter set built with the model.

## Dependencies

This parameter is enabled by **Load a parameter set from a file on the designated target file system**.

## Command-Line Information

**Parameter:** xPCOnTgtParamSetFileName

**Type:** string

**Value:** Any valid file name

**Default:** ''

## See Also

“xPC Target Options Configuration Parameters”

### Build COM objects from tagged signals/parameters

Enable build process to create a model-specific COM library file.

#### Settings

**Default:** off



On

Creates a model-specific COM library file, <model\_name>COMiface.dll.



Off

Does not create a model-specific COM library file.

#### Tip

Use the model-specific COM library file to create custom GUIs with Visual Basic or other tools that can use COM objects.

#### Command-Line Information

**Parameter:** xpcObjCom

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

#### See Also

“Creating the Target Application and Model-Specific COM Library”

## Generate CANape extensions

Enable target applications to generate data, such as that for A2L, for Vector CANape®.

### Settings

**Default:** off



On

Enables target applications to generate data, such as that for A2L, for Vector CANape.



Off

Does not enable target applications to generate data, such as that for A2L, for Vector CANape.

### Command-Line Information

**Parameter:** xPCGenerateASAP2

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### See Also

“Configuring the Vector CANape Device”

### Include model hierarchy on the target application

Includes the Simulink model hierarchy as part of the target application.

#### Settings

**Default:** off



On

Includes the model hierarchy as part of the target application.



Off

Excludes the model hierarchy from the target application.

#### Tips

Including the model hierarchy in the target application:

- Lets you connect to the target computer from xPC Target Explorer without being in the target application build folder.
- Can increase the size of the target application, depending on the size of the model.

#### Command-Line Information

**Parameter:** xPCGenerateXML

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

#### See Also

“Monitor Signals Using xPC Target Explorer”

## Enable Stateflow animation

Enables visualization of Stateflow® chart animation.

### Settings

**Default:** off



On

Enables visualization of Stateflow chart animation.



Off

Disables visualization of Stateflow chart animation.

### Command-Line Information

**Parameter:** xPCEnableSFAnimation

**Type:** string

**Value:** 'on' | 'off'

**Default:** 'off'

### See Also

“Animate Stateflow Charts Using Simulink External Mode”



# Target Computer Command-Line Interface Reference

---

## Target Computer Commands

### In this section...

“Introduction” on page 5-2

“Target Object Methods” on page 5-2

“Target Object Property Commands” on page 5-3

“Scope Object Methods” on page 5-5

“Scope Object Property Commands” on page 5-7

“Aliasing with Variable Commands” on page 5-8

### Introduction

You have a limited set of commands that you can use to work the target application after it has been loaded to the target computer, and to interface with the scopes for that application.

The target computer command-line interface enables you to work with target and scope objects in a limited capacity. Methods let you interact directly with the scope or target. Property commands let you work with target and scope properties. Variable commands let you alias target computer command-line interface commands to names of your choice.

Refer to “Target Computer Command-Line Interface” for a description of how to use these methods and commands.

### Target Object Methods

When you are using the target computer command-line interface, target object methods are limited to starting and stopping the target application.

The following table lists the syntax for the target commands that you can use on the target computer. The equivalent MATLAB syntax is shown in the right column, and the target object name `tg` is used as an example for the MATLAB methods. These methods assume that you have already loaded the target application onto the target computer.



<b>Target Computer Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
start	Start the target application currently loaded on the target computer. Syntax: start	tg.start or +tg
stop	Stop the target application currently running on the target computer. Syntax: stop	tg.stop or -tg
reboot	Reboot the target computer. Syntax: reboot	tg.reboot

## Target Object Property Commands

When you are using the target computer command-line interface, target object properties are limited to parameters, signals, stop time, and sample time. Note the difference between a parameter index (0, 1, . . .) and a parameter name (P0, P1, . . .).

The following table lists the syntax for the target commands that you can use to manipulate target object properties. The MATLAB equivalent syntax is shown in the right column, and the target object name `tg` is used as an example for the MATLAB methods.

<b>Target Computer Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
getpar	Display the value of a block parameter using the parameter index.  Syntax: getpar parameter_index	get(tg, 'parameter_name')
setpar	Change the value of a block parameter using the parameter index.  Syntax: setpar parameter_index = floating_point_number	set(tg, 'parameter_name', number)
stoptime	Enter a new stop time. Use inf to run the target application until you manually stop it or reset the target computer.  Syntax: stoptime = floating_point_number	tg.stoptime = number
sampletime	Enter a new sample time.  Syntax: sampletime = floating_point_number	tg.sampletime = number  set(tg, 'SampleTime', number)

Target Computer Command	Description and Syntax	MATLAB Equivalent
P#	Display or change the value of a block parameter. For example, P2 or P2=1.23e-4. Syntax: parameter_name or parameter_name = floating_point_number parameter_name is P0, P1, . . .	tg.getparam(parameter_index) tg.setparam(parameter_index, floating_point_number)
S#	Display the value of a signal. For example, S2. Syntax: signal_name signal_name is S0, S1, . . .	tg.getsignal(signal_index)

## Scope Object Methods

When using the target computer command-line interface, you use scope object methods to start a scope and add signal traces. Notice that the methods `addscope` and `remscope` are target object methods on the host computer, and notice the difference between a signal index (0, 1, . . .) and a signal name (S0, S1, . . .).

The following table lists the syntax for the target commands that you can use on the target computer. The MATLAB equivalent syntax is shown in the right column. The target object name `tg` and the scope object name `sc` are used as an example for the MATLAB methods.

<b>Target Computer Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
addscope	addscope scope_index addscope	tg.addscope(scope_index) tg.addscope
remscope	remscope scope_index remscope all	tg.remscope(scope_index) tg.remscope
startscope	startscope scope_index	sc.start or +sc
stopscope	stopscope scope_index	sc.stop or -sc
addsignal	addsignal scope_index = signal_index1, signal_index2, . . .	sc.addsignal(signal_index_vector)
remsignal	remsignal scope_index = signal_index1, signal_index2, . . .	sc.remsignal(signal_index_vector)
viewmode	Zoom in to one scope or zoom out to all scopes.  Syntax: viewmode scope_index or left-click the scope window  viewmode 'all' or right-click a scope window  Press the function key for the scope, and then press <b>V</b> to toggle viewmode.	tg.viewMode = scope_index tg.viewMode = 'all'
ylimit	ylimit scope_index ylimit scope_index = auto ylimit scope_index = num1, num2	sc.YLimit sc.YLimit='auto' sc.YLimit([num1 num2])
grid	grid scope_index on grid scope_index off	sc.Grid = on sc.Grid = off

## Scope Object Property Commands

When you use the target computer command-line interface, scope object properties are limited to those shown in the following table. Notice the difference between a scope index (0, 1, . . .) and the MATLAB variable name for the scope object on the host computer. The scope index is indicated in the top left corner of a scope window (SC0, SC1, . . .).

If a scope is running, you need to stop the scope before you can change a scope property.

The following table lists the syntax for the target commands that you can use on the target computer. The equivalent MATLAB syntax is shown in the right column, and the scope object name `sc` is used as an example for the MATLAB methods

Target Computer Command	MATLAB Equivalent
<code>numsamples scope_index = number</code>	<code>sc.NumSamples = number</code>
<code>decimation scope_index= number</code>	<code>sc.Decimation = number</code>
<code>scopemode scope_index = 0 or numerical, 1 or redraw, 2 or sliding, 3 or rolling</code>	<code>sc.Mode = 'numerical', 'redraw', 'sliding', 'rolling'</code>
<code>triggermode scope_index = 0, freerun, 1, software, 2, signal, 3, scope</code>	<code>sc.TriggerMode = 'freerun', 'software', 'signal', 'scope'</code>
<code>numprepostsamples scope_index = number</code>	<code>sc.NumPrePostSamples = number</code>
<code>triggersignal scope_index = signal_index</code>	<code>sc.TriggerSignal = signal_index</code>
<code>triggersample scope_index = number</code>	<code>sc.TriggerSample = number</code>
<code>triggerlevel scope_index = number</code>	<code>sc.TriggerLevel = number</code>
<code>triggerslope scope_index = 0, either, 1, rising, 2, falling</code>	<code>sc.TriggerSlope = 'Either', 'Rising', 'Falling'</code>

<b>Target Computer Command</b>	<b>MATLAB Equivalent</b>
triggerscope scope_index2 = scope_index1	sc.TriggerScope = scope_index1
triggerscopesample scope_index= integer	sc.TriggerScopeSample = integer
Press the function key for the scope, and then press S.	sc.trigger

## Aliasing with Variable Commands

The following table lists the syntax for the aliasing variable commands that you can use on the target computer. The MATLAB equivalent syntax is shown in the right column.

<b>Target Computer Command</b>	<b>Description and Syntax</b>	<b>MATLAB Equivalent</b>
setvar	Set a variable to a value. Later you can use that variable to do a macro expansion.  Syntax: setvar variable_name = target_pc_command  For example, you can type setvar aa=startscope 2, setvar bb=stopscope 2.	None
getvar	Display the value of a variable.  Syntax: getvar variable_name	None
delvar	Delete a variable.  Syntax: delvar variable_name	None
delallvar	Delete all variables.  Syntax: delallvar	None
showvar	Display a list of variables.  Syntax: showvar	None